UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Análisis de tráfico en Internet mediante la aplicación de redes neuronales convolucionales y recurrentes

Javier Aday Delgado Soto

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologias y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Análisis de tráfico en Internet mediante la aplicación de redes neuronales convolucionales y recurrentes

Autor: Javier Aday Delgado Soto
Tutor: Luis de Pedro Sánchez
Ponente: Jorge Enrique López de Vergara

junio 2021

Javier Aday Delgado Soto
Análisis de tráfico en Internet mediante la aplicación de redes neuronales convolucionales y recurrentes

Javier Aday Delgado Soto
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia

"Cualquier tecnología suficientemente avanzada es equivalente a la magia."

Arthur C. Clarke

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres y abuelos, por haber sido siempre un referente para mí, tanto a nivel académico como personal. Gracias por haberme introducido a este bonito mundo y por haberme apoyado en todo momento. Muchas gracias también a mis dos hermanos, veros crecer y ser mejores personas cada día, me ha hecho tener diariamente más ganas de crecer con vosotros.

También me gustaría agradecer todo su apoyo y ayuda a mis dos tutores, Luis y Jorge, por haberme aceptado y ayudado semana a semana a sacar adelante este proyecto.

Además, me gustaría darle las gracias a mi pareja, Rut, gracias por haber estado conmigo estos 4 años de esfuerzo constante y haber aguantado horas y horas de estudio y poco tiempo libre. Espero poder compartir muchas más metas contigo y seguir formando tan buen equipo. Muchas gracias también a su familia por todo su apoyo.

Por último, darles las gracias a todos los amigos que he ido haciendo a lo largo de estos 4 años. Compartir los malos y los buenos momentos con vosotros ha sido lo mejor que me han regalado estos años.

Resumen

Este Trabajo de Fin de Grado tiene como objetivo evaluar la viabilidad de aproximaciones basadas en Redes Convoluciones y Recurrentes para implementaciones en IDS mediante análisis de paquetes o de flujos. Este proyecto surge del creciente desarrollo e inversión en el sector de la Inteligencia Artificial y en el sector de la ciberseguridad, que en la sociedad actual ha adquirido una gran importancia y en un futuro seguirá esta tendencia.

Para el desarrollo de este TFG se ha hecho un estudio de las principales aproximaciones y se ha llegado a la conclusión de que existe la necesidad real de desarrollar un sistema que este basado en conjuntos de datos actuales y reales ya que la mayoría de estos estudios usan datos antiguos o sintéticos.

Por ello, se decide replicar el trabajo de Vinayakumar Ravi para así evaluar su investigación y la validez de su modelo propuesto. Como se quiere hacer una replicación de esta investigación se ha usado el mismo dataset, lo que nos sirve para evaluar también el conjunto de datos que se usa y sus deficiencias.

Después de evaluar estas deficiencias en el dataset y el modelo propuesto, se decide aplicar este mismo modelo a un conjunto de datos más actual y basado en datos quasi-reales para evaluar la eficacia del modelo.

El siguiente paso es hacer diferentes modificaciones a los datos de entrada, para ver si así se consiguen mejorar los resultados del modelo y, así, poder definir modelos más exactos a la hora de llevar a cabo esa clasificación.

Por último, se estudia la posibilidad de implementar diferentes arquitecturas de redes neuronales para aplicar en IDSs y las ventajas en inconvenientes que nos ofrece cada una de ellas.

Palabras clave

IDS, Clasificación de tráfico, Ciberseguridad, redes neuronales recurrentes, LSTM, redes neuronales convolucionales, paquetes, flujos, KDD-CUP 99' , CSE-CIC-IDS2018, Machine Learning, clasificación por votación

Abstract

This Degree Thesis aims to evaluate the viability of approaches based on Convolutional and Recurrent Neural Networks for IDS implementations by means of packet or flow analysis. This project arises from the growing development and investment in sectors like Artificial Intelligence and cybersecurity, which in today's society has acquired great importance and will continue this trend in the future.

The first step of this Thesis was a study of the main approaches that had been made and it was concluded that there is a real need to develop a system that is based on current and real data sets since most of these studies use old or synthetic data.

Therefore, it was decided to replicate Vinayakumar Ravi's work in order to evaluate his research and the validity of his proposed model. Since we want to replicate this research, we have used the same data set, which also serves to evaluate the data used and its shortcomings.

After evaluating these deficiencies in the data and the proposed model, it has been decided to apply this same model to a more recent dataset based on quasi-real data to evaluate the effectiveness of the model.

The next step is to make different modifications to the input data to see if this improves the model's results and thus be able to define more accurate models for classification.

Finally, the possibility of implementing different neural network architectures in IDSs and the advantages and disadvantages offered by each one of them are studied.

Keywords

IDS, traffic clasiffication, cybersecurity, recurrent neural networks, LSTM, convolutional neural networks, packets, flows, KDD-CUP 99' , CSE-CIC-IDS2018, Machine Learning, voting classification

Índice

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

1.1. Motivación y objetivos

En los últimos años, el sector de la Inteligencia Artificial ha crecido exponencialmente debido a razones como el aumento de la capacidad computacional, que nos permite ejecutar modelos más complejos. Esta Inteligencia Artificial ha permitido la optimización de gran cantidad de aplicaciones debido a su capacidad de extraer patrones de una gran cantidad de datos, cosa muy complicada para el ojo humano.

Esta característica de la Inteligencia Artificial hace que sea muy atractiva para resolver determinados problemas que representan un gran reto para el ser humano, entre ellos, la detección de ciberataques en tiempo real.

Al igual que la Inteligencia Artificial, la delincuencia en la red también ha crecido exponencialmente en los últimos años, cada día que pasa se llevan a cabo gran cantidad de actividades delictivas contra individuos o empresas que representan uno de los mayores peligros de la sociedad actual.

El sector de la ciberdelincuencia esta en constante cambio, los delincuentes van encontrando diferentes vulnerabilidades o modificando diferentes ataques, lo que hace que sea difícil mantenerse al corriente de todos los ciberataques. Para este fin, podemos hacer uso de la IA debido a su gran capacidad computacional y de detección de patrones.

Los sistemas de detección de intrusos en una red son los denominados IDS. Estos últimos años ha crecido la presencia de arquitecturas de IA en estos sistemas y es una aproximación de la que se espera mucho, aunque aun es una aproximación que necesita de mucha investigación para mostrarse eficiente.

Debido a esto último, en este trabajo fin de grado se explorará un modelo propuesto que usa una arquitectura CNN+LSTM para un IDS y se desarrollará este para actualizarlo y optimizarlo con la finalidad de continuar la investigación en este campo tan prometedor.

Los objetivos de este trabajo son los siguientes:

- Comprender la situación en la que se encuentra este campo y que investigaciones son necesarias para continuar desarrollándolo.
- Ser capaz de entender el funcionamiento de las redes CNN y LSTM para así comprender el modelo propuesto en la investigación.
- Discernir el preprocesado de datos aplicado en la investigación indicada para poder replicarla.
- Replicar satisfactoriamente la investigación comentada y hacer un análisis más extenso de sus resultados para saber si se han ocultado datos en esta o no.
- Actualizar el modelo propuesto de esta arquitectura para que este sea aplicable en la actualidad.
- Llevar a cabo una aproximación a un IDS basándose en el modelo actualizado.

1.2. Fases del trabajo

Este trabajo ha tenido una duración de 11 meses, desde julio de 2020 a junio de 2021. La estructuración del trabajo ha sido la siguiente:

- **Fase 1, Investigación**: En esta fase se hace un análisis del estado del arte de clasificadores de ciberataques basados en redes neuronales.
- **Fase 2, Selección**: Después de analizar gran cantidad de investigaciones, se selecciona una en concreto para su posterior replicación y estudio.
- **Fase 3, Replicación**: En esta fase se intenta hacer una replicación del algoritmo desarrollado en la investigación seleccionada, para así poder evaluarla y tener un punto de partida para futuros desarrollos.
- **Fase 4, Desarrollo**: En esta fase se desarrolla una aproximación a un IDS basándose en el modelo antes replicado y haciéndose las modificaciones que se crean convenientes para optimizar sus resultados.
- **Fase 5, Estudio de resultados**: En paralelo con la fase de replicación comienza esta otra fase donde, gracias a los conocimientos adquiridos en la investigación previa, se evalúa la eficacia y los resultados obtenidos del modelo desarrollado. Esta fase también se aplica a la fase 4. Ni la fase 3 ni la 4 concluirán si no se obtienen resultados positivos.
- **Fase 6, Redacción de la memoria**: En esta última fase se plasmarán los conocimientos obtenidos y los resultados generados en un documento para su posterior evaluación.

A continuación se muestra un diagrama de Gantt que representa la duración del proyecto y de cada fase además del tiempo total invertido en cada una de ellas:

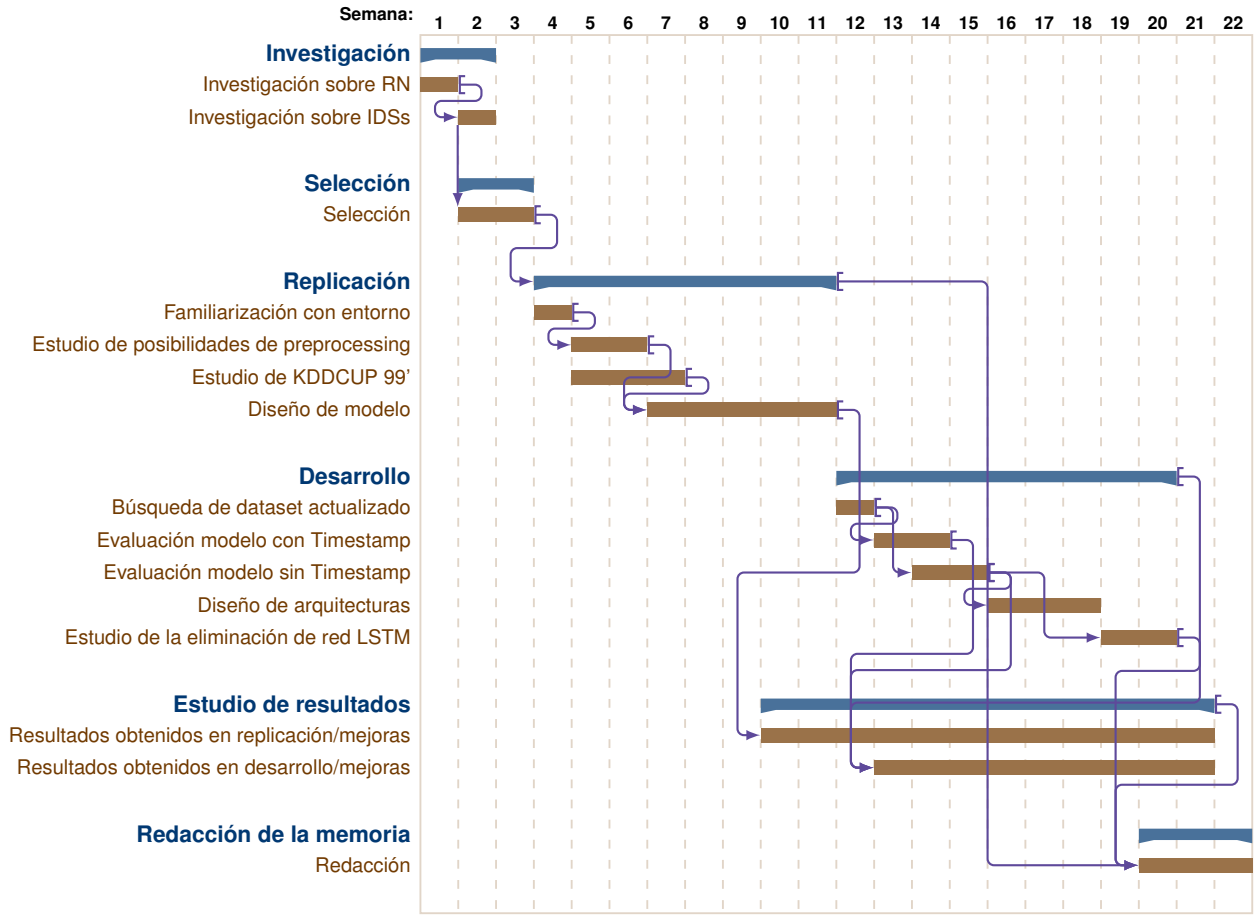

Figura 1.1: Diagrama de Gantt de desarrollo temporal del trabajo.

	Investigación	Selección	Replicación	Desarrollo	Resultados	Redacción
Tiempo(h)	35 h	10 h	92 h	137 h	70 h	30 h

Tabla 1.1: Tiempo invertido en cada fase

Tiempo Total
374 h

Tabla 1.2: Tiempo total invertido

1.3. Organización de la memoria

La organización de esta memoria es la siguiente:

- **Capítulo 2**: Análisis del estado del arte referente a redes neuronales y sus aplicaciones en detección de tráfico maligno.
- **Capítulo 3**: Aquí se muestra el entorno experimental, además del diseño utilizado y el desarrollo tanto para la replicación como para el desarrollo de un IDS.
- **Capítulo 4**: En este capítulo se expondrán los resultados obtenidos del trabajo y se comentarán estos.
- **Capítulo 5**: Por último, en ese capítulo se expondrán las conclusiones extraídas de este trabajo y posible trabajo futuro.

2

Estado del Arte

2.1. Introducción

Una parte esencial del desarrollo de este trabajo es el estudio de los conceptos que serán más relevantes en el diseño de este. En concreto los aspectos mas relevantes son las redes neuronales, los diferentes sistemas IDS y la unión de estos dos conceptos que representan los IDS basados en redes neuronales. Este capítulo es clave para ser capaz de discernir cual es el modelo más interesante para aplicarse en este trabajo.

2.2. Machine Learning

Machine Learning es un concepto que deriva de la Inteligencia Artificial (IA) [1]. En esencia, es un método de análisis de datos que está basado en la idea de que se pueden generar sistemas que sean capaces de tomar decisiones, identificar patrones demasiado complejos para el ojo humano y aprender de los datos sin necesidad de la intervención humana [1]. Se puede encontrar más información de este tema en el anexo B.

Una vez estudiadas las principales características del ML, se investiga concretamente el campo de las redes neuronales.

2.2.1. Redes Neuronales

Las redes neuronales son uno de los algoritmos más utilizados hoy en día y con mayores perspectivas de futuro [2]. La idea principal de este algoritmo es replicar en la medida de lo posible el funcionamiento de un cerebro. A día de hoy, debido a la limitada capacidad computacional, las redes neuronales son mucho menos potentes que el cerebro humano, las redes actuales se podrían corresponder con cerebros de animales como el de la abeja, que tiene 1 millón de neuronas. Esto representa un 0.001 % del número de neuronas de un cerebro humano [2]. Se pueden encontrar sus elementos básicos y otras explicaciones en el anexo C.

Las neuronas se agrupan por capas que se conectan entre sí para formar las redes neuronales. Las utilizadas en este trabajo estas descritas en el apéndice C.

2.3. IDSs

Un IDS es un sistema Hardware o Software cuya finalidad es monitorizar el tráfico en circulación a través de una red o sistema para detectar amenazas y alertar a la persona/personas encargada cuando se encuentre una de estas [3].Se puede encontrar más información acerca de los IDS en el anexo D.

2.3.1. IDSs basados en redes neuronales

Después del estudio realizado, se decide utilizar como base para este trabajo un diseño para un IDS basado en redes neuronales, por lo que en este apartado se comentarán algunos de los estudios más interesantes evaluados de este tema.

En primer estudio que se comentará se llama *"1D CNN based network intrusion detection with normalization on imbalanced data"* llevado a cabo por personal de las universidad Ihna y Gachon de Korea del Sur [4].

En esta primera investigación se proponen varias arquitecturas de redes convolucionales 1D para detección de intrusiones, diferenciadas entre sí por el número de capas de convolución usadas y la incorporación de redes LSTM recurrentes. En estos modelos se aplica normalización del batch para hacer que los resultados de train converjan antes y mejorar los resultados. Se utiliza el dataset UNSW_NB15 [5].

De este dataset se genera una versión balanceada y otra desbalanceada. En ambos casos el modelo que obtiene mejores resultados es el compuesto por una red CNN de 3 capas convolucionales, con un F1-Score de 91.59 % para la versión con datos balanceados y 88.46 % para datos desbalanceados. El modelo CNN+LSTM obtiene resultados similares pero ligeramente inferiores.

El segundo estudio relevante es *"DDoS attack detection and classification via Convolutional Neural Network (CNN)"* llevado a cabo por investigadores de la universidad de El Cairo, Egipto [6].

En este estudio se investiga la posibilidad de usar modelos basados en machine learning en centros de control de misión (MCC) para satélites y transportes espaciales. Su finalidad es la detección de ataques DoS/DDoS, para ello usan tanto el dataset NLS-KDD como uno propio generado un una red MCC simulada. Para la detección de este tipo de ataque se plantea usar una arquitectura basada en CNN con 2 capas convolucionales. El tráfico de entrada se convierte en imágenes 2D para los datasets.

Los resultados que obtiene este desarrollo para el modelo propuesto son, para el dataset generado

por lo investigadores, un accuracy de 99.33 % y para el NLS-KDD, 99.24 %. Por último, los investigadores evaluan otros modelos basados en ML como son los arboles de decisión, SVM, KNN y otro modelo de NN, pero todos ellos obtienen valores de accuracy menores al modelo CNN.

La tercera investigación que se estudiará será *"Applying Convolutional Neural Network for Network Intrusion Detection"* desarrollada por Vinayakumar Ravi y otros investigadores en la universidad Amrita de la India [7].

Este estudio investiga la posibilidad de desarrollar modelos ML para su aplicación en sistemas IDS. Para este fin se plantea el uso del dataset KDD-CUP 99' como punto de partida. Este dataset cuenta con una gran variedad de ataques, lo que es recomendable para un IDS. Sobre este dataset se evalúa la posibilidad de implantar diferentes modelos basados en CNN 1D. Se evalúan modelos de CNN puros de 1 a 3 capas de convolución y modelos híbridos de CNN junto con RNN como LSTM y GRU.

Para cada modelo se lleva a cabo tanto una clasificación binaria (benigno-maligno), como una clasificación multiclase. De entre todos los modelos aplicados, el que mejores resultados ofrece para clasificación multiclase es el del modelo híbrido CNN(3 capas) + LSTM, que obtiene un accuracy de 98.7 %.

2.4. Conclusiones

En este capítulo se ha revisado el estado actual tanto de las redes neuronales como de los planteamientos de detección de ciberamenzas basados en estas tecnologías, haciendo especial hincapié en los modelos basados tanto en CNN como en LSTM.

Este estudio comenzó por comprender el concepto de ML y reconocimiento de patrones, además de las diferentes aproximaciones aplicadas a su entrenamiento. Después de esto, se estudiaron diferentes modelos del ML pero se seleccionaron las CNN y RNN basándose en un criterio de gusto del alumno. Una vez comprendidos estos modelo, se estudió la viabilidad de estos para aplicaciones de detección de tráfico.

Un aspecto común y preocupante de todas las investigaciones es el hecho de que en todas ellas se usa el parámetro accuracy para representar la calidad del modelo cuando esta demostrado que esta métrica no es la más adecuada debido a que puede inducir a error como se comenta en el apéndice A.1.

3

Diseño y Desarrollo

3.1. Introducción

En este capítulo se comentarán las diferentes fases del diseño del algoritmo y las variables que han influenciado en su desarrollo.

Este trabajo tiene 4 partes diferenciadas, cada una dependiente de la anterior salvo la última. En este apartado se irán comentando todas, los diferentes apartados son:

1.– **Replicación de investigación de Vinayakumar Ravi** [7].

2.– **Adaptación del algoritmo a CSE-CIC-IDS2018**.

3.– **Diseño de 2 arquitecturas para detección completa**.

4.– **Eliminación de red LSTM**.

Los diagramas de desarrollo de los diferentes apartados son los siguientes:

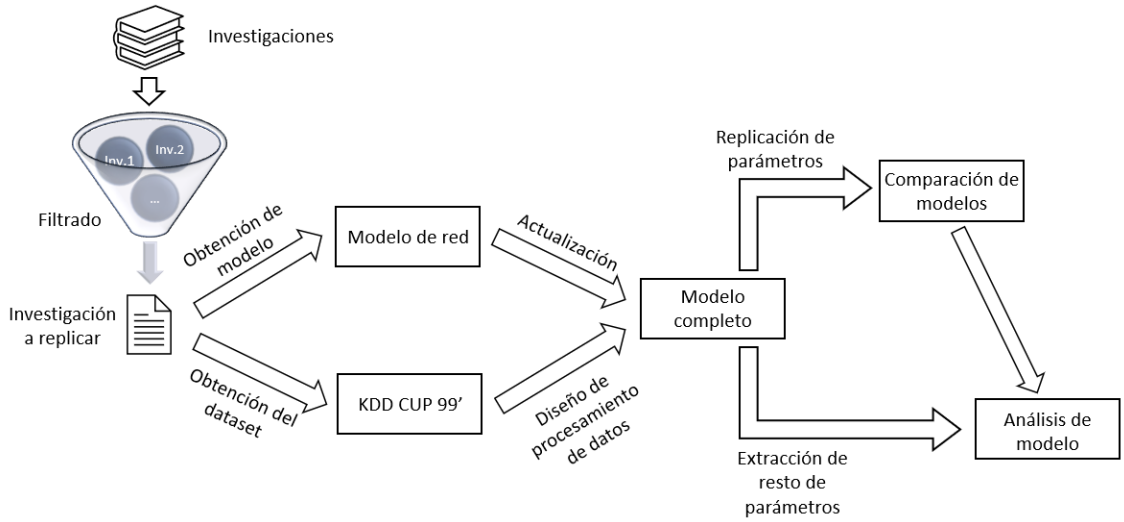

Figura 3.1: Desarrollo de apartado 1

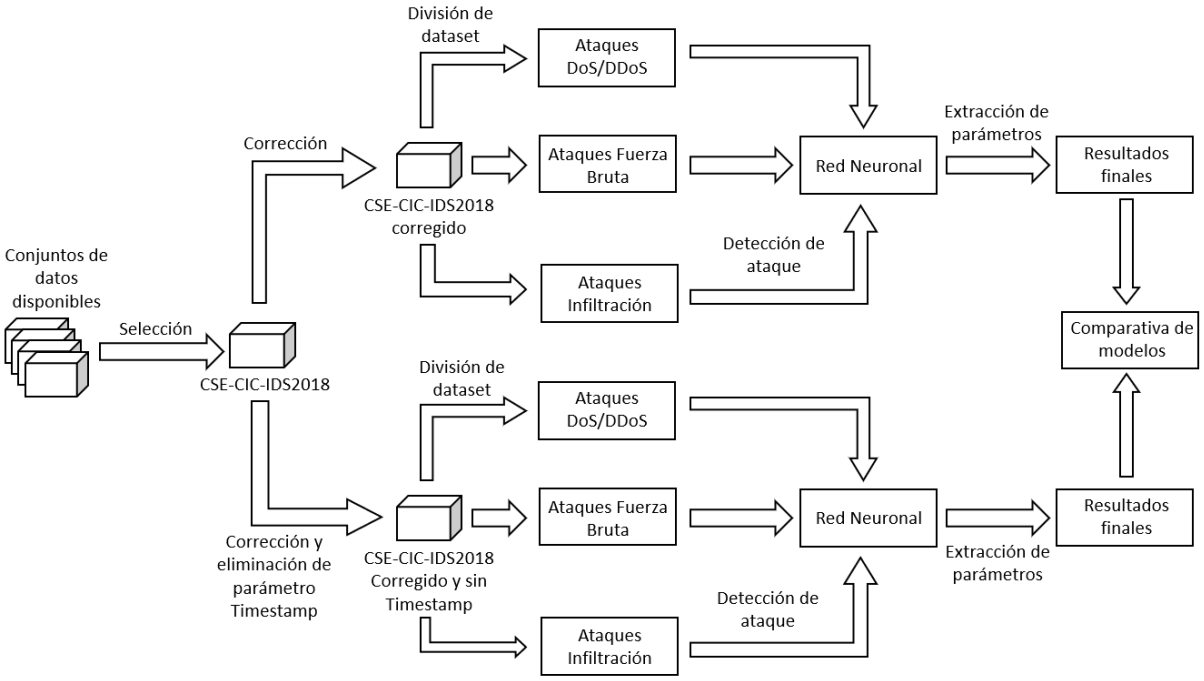

Figura 3.2: Desarrollo de apartado 2

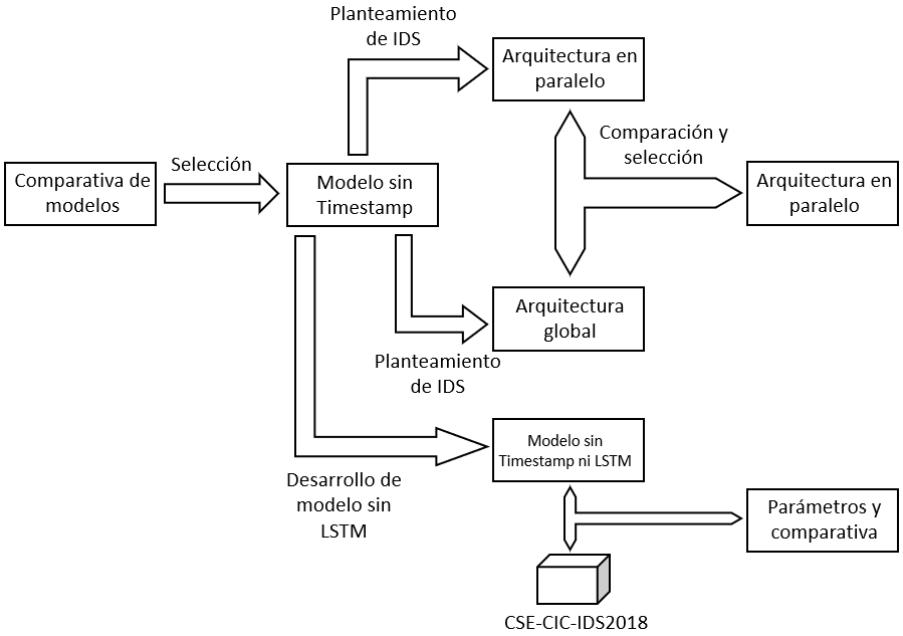

Figura 3.3: Desarrollo de apartados 3 y 4

3.2. Entorno

3.2.1. Conjuntos de datos

KDD-CUP 99'

Este es el conjunto de datos usado por Vinayakumar Ravi en [7]. Se ha decidido usar este conjunto de datos porque nos permitirá evaluar el modelo desarrollado por Vinayakumar.

Este dataset fue desarrollado por la *University of California, Irvine* [8] en el año 1999 a partir de datos procesados del IDS desarrollado en 1998 por DARPA [9]. Se creó con la idea de ser aplicado en la competición "The Fifth International Knowledge Discovery and Data Mining Tools Competition" [10].

Este dataset consiste aproximadamente de 4,900,000 vectores que representan cada uno una conexión. Cada vector contiene 41 características [11] que se pueden dividir en 3 grupos.

- **Básicas**: representan los parámetros básicos que se pueden obtener de una conexión TCP/IP.
- **Tráfico**: representan parámetros obtenidos con respecto a ventanas de tiempo de 2 segundos. Se dividen en 2 grupos.
    - *Mismo host destino*
    - *Mismo servicio*
- **Contenido**: sirven para una mejor detección de ataques que no tengan patrones temporales de intrusión. Representan parámetros propios del paquete.

Por último, cada vector contiene también el tipo de tráfico al que pertenece, en este dataset podemos encontrar los siguientes "Labels":

- **Normal**: tráfico benigno.
- **DoS**: Smurf, Back, Teardrop, Land, Neptune y Pod.
- **Probe**: Nmapr, Portsweep, Satan e IPsweep.
- **R2L**: Multihop, Spy, Ftpwrite, Phf, Warezclient, Warezmaster, Imap y Guesspasswd.
- **U2R**: Rootkit, Perl, Bufferoverflow y Load module.

CSE-CIC-IDS2018

Este dataset [12] [13] se ha usado para evaluar la viabilidad de esta arquitectura a la hora de detectar tráfico más actual. Los datos de este conjunto están organizados de manera diferente que los del KDD-CUP 99' por lo que será necesaria una modificación en el código.

No es un volcado de paquetes de tráfico como el anterior, ha sido creado como un proyecto colaborativo entre *Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC)*, con la finalidad de hacer frente a los principales problemas que enfrenta este sector: bases de datos privadas no accesibles para la mayoría y datos antiguos o mal distribuidos estadísticamente.

Para la generación de este dataset se han usado 2 redes, una haciendo la labor de agresor y la otra de víctima. La agresora cuenta con 50 maquinas y la víctima se asemeja a una red corporativa con 5 departamentos, 420 maquinas y 30 servidores.

Esta compuesto por flujos y no por paquetes. Cada flujo esta definido por 80 parámetros extraídos a través de la generación de flujos realizada en CIC-Flowmeter. Estos parámetros comprenden desde análisis estadístico de los paquetes dentro del flujo hasta variables técnicas de estudio de paquetes.

En cuanto a tipos de tráfico disponibles, en este conjunto de datos podemos encontrar:

- **Normal**: tráfico benigno.
- **DoS**: Slowloris.
- **DDoS**: Hulk, LOIC, Goldeneye y Slowhttptest.
- **Infiltración**: Usando DropBox y Nmap - Portscan.
- **Botnets**: Zeus (troyano) y Ares.
- **Fuerza Bruta**: usando FTP y SSH.

3.2.2. Herramientas

Jupyter Notebook

Jupyter notebook [14] ha sido el IDE elegido para el desarrollo de gran parte de este trabajo. Es un IDE muy potente y versátil, capaz de entender gran cantidad de lenguajes y librerías. Una de las principales razones por las que se ha elegido este IDE es el hecho de que permite la ejecución de código por partes, lo que es una gran ayuda a la hora de detectar fallos o mejoras.

Al usar Jupyter Notebook los modelos han corrido sobre el ordenador del alumno, este ordenador consta de un procesador amd ryzen 3 5600 y una GPU Shappire Radeon 580. Se ha investigado la posibilidad de optimizar el funcionamiento del modelo haciendo que corriese sobre la GPU pero las soluciones encontradas eran poco viables. También se ha evaluado la opción de usar otro IDE que permitiera el acceso a GPUs o TPUs como Google Collab pero se desestimo la idea debido a problemas al manejar el gran tamaño de datos necesario para entrenar el modelo.

Se tomó la decisión de usar Jupter Notebook después de hacer la comparativa mostrada en la siguiente figura:

Anaconda

Anaconda [15] es una de las distribuciones de software libre más conocidas del sector de ciencia de datos e IA. Su desarrollo ha sido orientado al tratamiento de grandes volúmenes de datos, cómputos científicos y análisis predictivos.

Esta distribución utiliza su propio gestor de paquetes llamado *conda* y su propia terminal llamada

	Jupyter Notebook	Anaconda Spyder	VS Code	Matlab
Soporte Windows	✓	✓	✓	✓
Facilidad de uso	✓	✓	✓	✓
Uso gratuito	✓	✓	✓	~ (Licencia Universitaria)
Exploración modular de datos/código	✓	X	X	X
Librerías de ML disponibles	✓	✓	✓	~ (Menos facilidad de uso)
Linting	✓	✓	✓	✓
Decisión final	✓	X	X	X

Figura 3.4: Comparativa de entornos para la realización del trabajo

Anaconda Prompt. Desde este gestor de archivos se pueden ir descargando librerías como Keras y Tensorflow, necesarias para el desarrollo del trabajo.Se ha ejecutado Jupyter Notebook dentro de Anaconda para así aprovechar los beneficios de ambos servicios.

Tensorflow y Keras

Tensorflow [16] es la librería de Aprendizaje Automático más relevante. Es una de gran plataforma para diseñar y evaluar modelos de redes neuronales, su diseño brinda al usuario final una gran capacidad de cómputo que permite realizar las operaciones rápidamente (siempre dependiendo del HW sobre el que se esté ejecutando). Tensorflow puede trabajar sobre gran cantidad de dispositivos, pero su funcionamiento sobre GPUs está ligeramente restringido a las de la familia NVIDIA, ya que Tensorflow funciona usando núcleos CUDA no disponibles en GPUs AMD. Para poder usar Tensorflow sobre GPUs AMD se debe utilizar la plataforma ROCm (desarrollada por AMD) [17]. El problema de esta plataforma es que está muy estrechamente ligada con Linux y aún no se tienen planes de migrarla a Windows. Otro elemento estrechamente ligado a Tensorflow son las TPUs, estas son una construcción ASIC diseñadas exclusivamente para el procesamiento de tensores, lo que los convierte en un acelerador para modelos de IA, ya que cuentan con un alto throughput de elementos aritméticos de precisión baja. Estas TPUs están especialmente diseñadas para la ejecución de modelos y no para su entrenamiento.

Keras [18] es una librería de alto nivel que fue desarrollada en Python para reducir los tiempos de creación de redes neuronales e incrementar la facilidad de uso y comprensión tanto de las librerías sobre las que funciona como de los modelos creados. Es capaz de trabajar sobre librerías como Theano o Tensorflow. La función principal de Keras es ser la API entre el usuario y las librerías de cómputo, además de soportar programación en muchos elementos HW.

CICFlowmeter

CICFlowmeter [19] es un analizador y generador de flujos desarrollado por *Canadian Institute for Cybersecurity (CIC) [20] y Arash Habibi Lashkari [21].* Está disponible para Linux en [22].

Con este programa se generó el dataset CSE-CIC-IDS2018. Durante pruebas realizadas por el alumno para revisar y entender su funcionamiento, se descubrió un error de este analizador a la hora de detectar tráfico ARP, ya que este era confundido con tráfico TCP [23].

Kali Linux

Kali Linux [24] es una distribución open source Linux basada en Debian desarrollada por Offensive Security. En esta distribución se encuentran la gran mayoría de herramientas necesarias para hacer pruebas de pentesting, análisis forense, etc. Cuenta con mas de 300 herramientas que permiten replicar los diferentes ataques usados para generar los datasets.

En este caso se usó Metasploit para generar un archivo PDF con un payload que ejecutaba una backdoor en el ordenador con Windows Vista de la víctima al abrir el documento con Adobe Acrobat 9 al sobrecargar un buffer. Esta es la replicación de uno de los ataques usados para generar los datos de infiltración del dataset CSE-CIC-IDS2018 con la finalidad de luego usar CICFlowmeter para comprender el funcionamiento de ambos.

3.3. Replicación de la investigación de Vinayakumar R.

Este apartado comienza después de la fase de estudio del arte. El alumno revisó las investigaciones más relevantes publicadas y seleccionó esta basándose en ciertos criterios:

	Estudio 1	Estudio 2	Estudio 3
Fecha de publicación	19 Feb 2020	10 Dic 2019	4 Dic 2017
Calidad del documento y de la investigación	10/10	9/10	10/10
Participación del autor en otros desarrollos del sector	3.33/10	4/10	6/10
Importancia de la investigación	11	10	199
Arquitectura planteada	CNN 3 capas	CNN 2 capas	CNN(3 capas + LSTM)
Datos utilizados	UNSW_NB15	Datos propios/ NLS KDD	KDD-CUP 99'
Resultados obtenidos	91.2% (F1)	99.24% / 99.3% (Acc)	98.7% (Acc)

Figura 3.5: Comparativa de investigaciones

Después de evaluar todos estos parámetros, la investigación más interesante ha sido la comentada anteriormente [7]. En el apéndice E se comenta el modelo seleccionado procedente de esta investiga-

ción.

Para empezar con la reproducción de la investigación, se empezó por intentar reproducir su arquitectura. Este paso resultó bastante sencillo, ya que es fácil llegar al repositorio de GitHub de Vinayakumar, donde tiene publicados los códigos de sus arquitecturas [25].

El primer problema al que se enfrentó el alumno en la realización de este TFG es que Vinayakumar solo tiene publicados los códigos de creación de las arquitecturas, no tiene publicado el tratamiento de datos que hizo para adaptar el dataset. Este trabajo lo realizó el estudiante.

Este primer problema es generado por el hecho de que, dentro del dataset KDD CUP 99', algunos de sus parámetros están definidos como elementos no numéricos, por lo tanto deben ser convertidos a elementos numéricos para poder ser tratados por la red.

Esta conversión a valores numéricos se ha hecho con sklearn [3] [26], ya que esta librería ofrece recursos muy potentes para el tratamiento de datos, además de una gran comunidad y documentos explicativos. Se han probado diferentes aproximaciones para esta conversión como son *OneHotEncoder* [27] y *LabelEncoder* [28]. Después de revisar las diferentes salidas obtenidas se eligió *LabelEncoder* como aproximación final debido a que era con la conversión con la que mejor se mantenían las relaciones de diferencia entre paquetes.

Esta conversión solo ha sido necesaria aplicarla a determinadas columnas como son la columna 2 (Tipo de protocolo de conexión) , 3 (servicio), 4 (flag) y 42 (Tipo de tráfico). A continuación se incluye un ejemplo de la conversión aplicada.

Código 3.1: Conversión de tipo de columna 42

```
kdd_cup_99[41] =lb_make.fit_transform(kdd_cup_99[41])
```

La conversión realizada por LabelEncoder es simplemente obtener todas las posibilidades para ese parámetro disponibles dentro del dataset (definido como N) y luego sustituir ese mismo parámetro de cada paquete por valores comprendidos entre 0 y N-1, dependiendo del orden de aparición.

Una vez aplicado el cambio de tipos de dato a las variables problemáticas, el último paso del procesamiento de datos sería dividir los datos entre conjunto de test y conjunto de entrenamiento. Para esta labor también se decide usar sklearn [3] y su función *train_test_split* [29].

Código 3.2: Generación de datos de test y train

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y)
```

Con esta aproximación hacemos un ligero shuffle de los datos y además dividimos los datos otorgando un 25 % de estos a test y un 75 % a entrenamiento, un porcentaje común para entrenamiento de modelos. Hacer un shuffle de los datos tiene sentido para este dataset ya que si se revisan los datos es fácil darse cuenta de que los datos no están mezclados. Los diferentes tipos de tráfico están

ligeramente agrupados, lo que haría peor tanto la separación de datos para test-entrenamiento como el propio entrenamiento del modelo.

Al separar el conjunto de datos lo único que nos quedaría por hacer es normalizar estos datos para un mejor procesamiento de la arquitectura planteada. Una vez normalizados y convertidos a array, el procesamiento previo de datos ha terminado y están listos para entrenar/evaluar el modelo.

Por último, y después de haber explicado todos los pasos previos de tratamiento de datos y el modelo comentado en el apéndice E, se ejecuta este, los resultados de este pueden ser vistos en 4.3.1.

3.4. Adaptación a CSE-CIC-IDS2018

Después de la replicación de la investigación, se quiso estudiar la viabilidad de la aproximación sobre unos datos más actuales. Para esta labor se optó por mantener el modelo comentado anteriormente con sutiles modificaciones y realizar un cambio de los datos de entrada.

3.4.1. Modificaciones

El cambio de los datos de entrada supuso ligeras modificaciones en el modelo, no en la arquitectura si no en el tamaño de algunas capas como la de entrada ya que, al tener ahora una longitud de array de entrada de 79 elementos, esta capa debe tener tener también tamaño 79.

Otra de las modificaciones necesarias se debe hacer a la salida, ya que al cambiar los datos, cambiamos los tipos de tráfico, por lo tanto, tenemos un número total de tipos diferente. Este cambio pasa a ser muy relevante ya que, debido a la estructura del nuevo conjunto de datos (no está todo en un mismo conjunto si no que los datos están divididos en varios subconjuntos), se debe modificar el tamaño de la salida para cada uno de estos. Debido a este problema se aplica una optimización al modelo haciendo que esta última capa se redimensione automáticamente dependiendo del número de tipos de tráfico en los datos de entrada.

La última modificación del algoritmo diseñado en el apartado anterior es referente al tratamiento de los datos de entrada, al ser estos diferentes de los del apartado anterior, su tratamiento difiere ligeramente.

Dentro de las modificaciones aplicadas al pre-procesamiento de los datos, la primera que se debe aplicar debe ser unificar todos los tipos de datos de los parámetros, debido a que este dataset se compone en gran medida de variables estadísticas, se opta por un tipo de dato como float64, excluyendo la columna del tipo de tráfico que seguirá siendo la última.

La segunda modificación a aplicar es la corrección de estos datos ya que, es posible encontrar filas

erróneas que contengan valores no numéricos de los parámetros replicando estos valores los de la definición de cada parámetro. Este error es sencillo de eliminar sabiendo que todas las líneas erróneas contienen los mismos valores:

Código 3.3: Eliminación de datos mal introducidos

```
dataset = dataset.drop(dataset[dataset['Dst Port']=='Dst Port'].index)
```

La última modificación necesaria no es referente al algoritmo si no a la forma de comprender los datos de entrada y, por ende el funcionamiento del modelo. Los datos de entrada ya no son extraídos de paquetes, si no que ahora lo son del flujo. Por lo tanto, ahora tendremos un modelo que, en vez de basarse en la individualidad de las conexiones, es capaz de tomar decisiones basadas en una aproximación menos especifica en cuanto a la información transmitida y más basada en factores generales de las conexiones. Esto generará mayor robustez en el modelo ya que será capaz de adaptarse mejor a los cambios y avances de los diferentes ciberataques.

Una vez comentada la adaptación del algoritmo y el modelo al nuevo conjunto de datos, antes de mostrar los resultados obtenidos, es interesante comentar la evaluación del conjunto de datos.

Estudio del dataset

Lo más destacado de este conjunto de conjuntos de datos son los porcentajes de cada tipo de tráfico, obteniendo los siguientes porcentajes de [30]:

Tipo de tráfico	%	Tipo de tráfico	%	Tipo de tráfico	%
Benigno	83.07 %	**DDoS**	4.031 %	**Fuerza bruta**	2.347 %
Botnet	1.763 %	**Infiltración**	0.997 %	**Ataque Web**	0.006 %

Tabla 3.1: Porcentajes de CSE-CIC-IDS2018

Al ver esta tabla se observa claramente que se esta tratando con un dataset con un gran desbalanceo entre clases. Esto es entendible ya que este conjunto de datos intenta replicar el funcionamiento real de una red corporativa, por lo que tiene sentido tener un gran desbalanceo.

Otro hallazgo publicado en [30] es el hecho de que la mayoría de las investigaciones tienen unos resultados inusualmente altos, cosa que según este estudio es debido al overfitting. El modelo planteado anteriormente ya tenía una capa dedicada a disminuir los errores producidos por el overfitting, por lo que se seguirá con este modelo y ,si los resultados obtenidos son inusualmente altos, se aumentará el ratio de Dropout.

Respecto al problema del desbalanceo de las clases, se han evaluado diferentes posibilidades para intentar disminuir el impacto de este hecho, pero ,al final, se ha decidido no modificar el conjunto de datos ya que la principal finalidad de este TFG es crear una aproximación a un IDS que sea aplicable

y realista, por esto mismo, el alumno cree que modificar el conjunto de datos haría que este perdiera el factor que lo diferencia respecto del otro dataset, el realismo.

Aun así, las diferentes posibilidades que se tuvieron en cuenta en caso de que se decidiese solventar el problema del desbalanceo fueron:

- **Undersampling**
- **Oversampling**
- **Muestreo sintético**

La primera solución evaluada ha sido el **Muestreo sintético**, intentando añadir muestras de las clases con menor porcentaje. El desarrollo se hizo con ataques de infiltración.

Lo que, de hizo fue replicar las condiciones comentadas en la página oficial de CSE-CIC-IDS2018 [12] para los ataques de infiltración y usando Wireshark agrupar los paquetes tanto del ciberataque como del tráfico benigno paralelo para un posterior procesamiento en CICFlowmeter y por último ser añadido al conjunto de datos. El sistema de generación y procesamiento de datos será el mostrado en M.1.

A la hora de poner este sistema en marcha hay que destacar que para ello fue necesario un estudio de Kali Linux [24] y de las herramientas que este sistema brinda. Como se quería obtener flujos de ataques de infiltración y estos debían ser similares a los ya presentes en el dataset, desde Metasploit se generó un archivo PDF denominado .evil.pdf"que llevaba oculto un payload llamado *adobe_pdf_embedded_exe* [31].

Al abrir un archivo con este payload en Adobe Acrobat 9, se sobrecarga un buffer y esto permite ejecutar el payload que genera una backdoor desde ese dispositivo Windows Vista al dispositivo del atacante, que debe estar escuchando desde la terminal propia de Metasploit u otra equivalente. Una vez se ha conseguido ejecutar el backdoor y ambos dispositivos están conectados, el atacante tiene acceso a todas las direcciones dentro del ordenador de la víctima y al interior de la red en la que se encuentre el dispositivo afectado. Esto permite al atacante hacer un mapping interno de la red o generar cualquier otro tipo de perjuicio a la víctima.

En el caso práctico realizado por el alumno, este capturaba el tráfico de esta conexión entre ambos dispositivos con Wireshark [32] para luego tratarlos en CICFlowmeter.

Durante el tratamiento de datos de CICFlowmeter, el estudiante se dió cuenta de un potencial error de esta aplicación. Como se comentó en el siguiente issue del repositorio oficial en Github [23], el error se debe a que CICFlowmeter confunde una serie de bits que representan parámetros del tráfico ARP por las direcciones DSTIP y SRCIP, lo que genera un grave error en los datos ya que se confunden 2 parámetros muy relevantes.

Teniendo en cuenta este error y el hecho de que para conseguir corregir el desbalanceo de clases

seria necesario obtener cientos de miles de flujos de infiltración. Como estos requieren de la participación activa del estudiante para ser generados ya que se debe generar tráfico a través de la backdoor ejecutando diferentes instrucciones desde el terminal del atacante, y tráfico benigno paralelo se ha considerado esta aproximación para corregir el desbalanceo inviable.

La siguiente posible solución evaluada fue el **Undersampling**. Esta solución se desechó desde un inicio debido a que sería convertir un dataset, que simula el tiempo real, en algo para lo que no fue diseñado, ya que al eliminar aleatoriamente gran parte de las muestras de los tipos con mayor porcentaje, el dataset deja de reproducir el flujo real de tráfico en una red.

A la misma resolución se llegó cuando se evaluó el **Oversampling**. Los inconvenientes superaban a los beneficios ya que replicar muestras hace el dataset menos realista por lo que también se decidió no aplicar esta aproximación.

Al decidir no modificar el conjunto de datos nos quedamos con un dataset enormemente desbalanceado. Pero esto es compensado por el hecho de que representa tráfico real y de que las mediciones de porcentajes realizadas no son del todo ciertas, ya que son tomadas respecto al conjunto total. La propia estructura del dataset ayuda a reducir el efecto del desbalanceamiento ya que ,al tener divididos los ataques en diferentes subconjuntos, varios ciberataques no coexisten en el mismo .csv por lo que el desbalanceo principalmente vendrá dado por el trafico benigno. Debido a esto el desbalanceo se disminuye, ya que, el tráfico benigno incluido en cada uno de los subconjuntos es solo una parte del total y se ve reducido drásticamente, mientras que el tráfico maligno no se ve reducido por el mismo factor lo que mejora la relación de tamaño entre ellos. Además no se debe olvidar la capa de Dropout que posee el modelo con la que se puede corregir el posible overfitting producido por este desbalanceo.

3.4.2. Evaluaciones y desarrollo

El primer paso en la evaluación de los datos fue hacer un análisis por cada subconjunto de datos para evaluar la capacidad de detección del modelo de cada uno de los ciberataques y así tener una primera aproximación de la aplicación óptima de un posible IDS.

Al estudiar los resultados obtenidos, se observo un grave error en el algoritmo procedente de las funciones de generación de matrices de confusión usadas para evaluar el modelo. Este error se debe a que esta función va incluyendo tipos de trafico en las filas de la matriz de confusión a medida que va encontrando estos tráficos en los archivos "predicted.txt". Esto hace que si un ataque en la etapa de preprocessing se le ha asignado el valor 7 porque era el séptimo valor que aparecía pero en "predicted.txt"se encuentra en las primera posiciones debido a algún error, a la hora de representar la matriz de confusión se colocará en la segunda fila que debería corresponder al tipo de tráfico con valor asignado 1. Este desorden hace que se deba modificar esta parte del código y se optimice la generación de matrices de confusión para representarlas de una forma más visual usando scikitplot.

El procedimiento seguido para arreglar este error es muy simple, la funcion LabelEnconder nos permite hacer la transforma inversa de esta y devuelve las clases presentes en el dataset de manera ordenada por lo que valdría con hacer la transformada inversa de Y.

Con estas modificaciones y la función *scikitplot.metrics.plot_confusion_matrix* [33], se consigue arreglar el error y mejorar la representación de la salida.

Una vez evaluados los resultados de cada uno de los subconjuntos, y después de un segundo análisis por parte del alumno de los datos de entrada, se decide eliminar la columna de TimeStamp y volver a realizar las pruebas. Esta decisión se debe a que esta columna representaba el momento exacto del flujo, con fecha incluida, lo que es un mal parámetro de entrenamiento si se quiere generar un IDS aplicable en multitud de circunstancias ya que genera un sesgo poco beneficioso a la hora de evaluar un flujo de otro momento horario.

Otro elemento a tener en cuenta a la hora de la toma de esta decisión es que el modelo ya cuenta con una red LSTM capaz de gestionar dependencias temporales, por lo tanto, la eliminación de este parámetro no representa ningún problema, y puede ser beneficiosa para una futura implementación.

Después de evaluar los resultados caso por caso, se ha considerado una buena idea evaluar también la capacidad del modelo de analizar el tráfico y a la vez ser capaz de distinguir entre diferentes variantes del mismo tipo de ataque. Para esto se han generado nuevos conjuntos de datos agrupando los ataques en 3 clases:

- **DoS/DDoS**
- **Fuerza bruta**
- **Infiltración**

Después de crear estos 3 conjuntos de datos, se vuelve a entrenar el modelo con cada uno de ellos y a evaluar los resultados obtenidos. Este paso sera un pilar fundamental a la hora de crear un IDS que funcione con redes en paralelo.

3.4.3. Aproximación a IDS

A raíz de las evaluaciones comentadas previamente, se ha decidido crear un algoritmo que implemente un IDS basado en los modelos anteriores. A la hora de diseñar este IDS, se ha optado por evaluar 2 tipos de arquitectura diferentes, en paralelo y global.

Arquitectura en paralelo

Esta arquitectura se ha diseñado para poder ser escalada a sistemas de gran capacidad de procesamiento. Las principales ventajas de esta arquitectura son el hecho de que es más exacta a la hora de detectar amenazas debido a la especialización de cada una de las redes y el hecho de que se ejecute

computo en paralelo para reducir los tiempos de procesamiento de flujos.

Como se comentó en el apartado anterior, esta arquitectura paralelo cuenta con 3 redes neuronales similares especializadas cada una en la detección de un tipo de amenaza diferente, la primera de ellas en ataques DoS y DDoS, la segunda en ataques de fuerza bruta y la última en infiltraciones.

Todas las redes tienen la misma forma, mismos parámetros de entrada y capas intermedias idénticas. Su única diferencia es el tamaño de la capa de salida ya que este parámetro es dependiente del número de tipos de tráfico y estos son diferentes para ataque.

El algoritmo que se ha implementado de manera que la primera parte de este sea común y se pueda aplicar a todos los detectores. Esta primera parte consiste en el tratamiento de los datos, que se hace de forma idéntica a como se hacia para apartados pasados.

El siguiente paso en la ejecución del algoritmo sería la creación de las 3 redes neuronales teniendo en cuenta las diferentes salidas para cada una. Una vez hecho esto, se puede ejecutar en paralelo cada modelo, para lograr esto, se ha usado la biblioteca *multiprocessing* [34]. En concreto se ha utilizado el trozo de código encontrado en L.1.

En este fragmento de código podemos ver que la función *multiprocessing,Process()* para generar los procesos donde se ejecutarán cada una de las redes. En cada proceso se ejecuta la función *Prediccion()* (que se explicará más adelante) y a esta función se le pasan como argumento 3 parámetros:

1.– **i**: representa a la red que se quiere usar, se estudiarán sus valores más adelante.

2.– **input_NN**: representa los datos de entrada de las redes.

3.– **networks[i]**: Es un array de strings donde cada uno representa una ruta absoluta al archivo donde se han guardado los pesos de cada una de las redes neuronales.

Después de comenzar cada uno de estos 3 procesos, se añaden al array "procesos"para poder controlar el final de cada uno y hacer que no se siga ejecutando código hasta que todos acaben. Esto es especialmente relevante debido a las limitaciones de la librería multiprocessing en Windows, estas limitaciones se comentarán más adelante.

La función *Predicción* llamada por cada proceso se puede ver en L.1. En esta función es posible observar el uso de cada uno de los parámetros comentados anteriormente. Se pueden identificar tres grupos idénticos de código, esto se debe a que en esta función se han aunado todas las redes por un motivo de eficiencia de código. Se observa que para acceder a un segmento de código o a otro nos basamos en el valor .ºption"que se corresponde con ï.[en] la llamada, como se ha comentado antes, este parámetro sirve para seleccionar una red u otra. Lo siguiente que hace el código mostrado es cargar el modelo desde la ruta pasada, por esto es extremadamente importante que el orden de las rutas en la variable "networks"se corresponda directamente con los números asociados a cada red en “option". Una vez cargado el modelo en la red se procede a hacer la predicción de los datos de entrada.

Al ejecutarse las redes neuronales en diferentes procesos para implementarlas en paralelo, es

necesario desarrollar un sistema de comunicación entre procesos para permitir que el proceso padre conozca los resultados obtenidos en las diferentes predicciones. Para implementar esta comunicación se ha decidido usar archivos de texto externo ya que representan la forma más básica de comunicación y nos permiten tener un punto de acceso para posibles revisiones del funcionamiento del sistema.

Sobre el funcionamiento de estos archivos solo se quiere destacar una cosa más, al estar este sistema diseñado para hacer pruebas, estos archivos se borran cada vez que se ejecuta la función predicción para que así datos anteriores no afecten a los resultados.

Después de escribir en los archivos los procesos hijos, el proceso padre lee estos archivos y mediante un sencillo sistema de votación se decide a que tipo de tráfico corresponde cada flujo evaluado. El sistema de votación se muestra en la tabla 3.2 (X representa predicción de ataque):

Tipo1	Tipo2	Tipo3	Salida
			Benigno
X			Tipo1
X	X		Tipo1-Tipo2
X		X	Tipo1-Tipo3
	X	X	Tipo2-Tipo3
X	X	X	Tipo1-Tipo2-Tipo3

Tabla 3.2: Sistema de votación

Se ha encontrado un gran problema a la hora de implementar esta modificación para diseñar el sistema en paralelo. Este problema nace en la librería *multiprocessing* debido a su implementación en Windows. A la hora de comenzar un proceso en Windows, esta librería no usa el **fork** por defecto (que generaría un proceso hijo idéntico al padre con los mismos recursos y desde ese punto en el código), si no que usa **spawn** (que ejecuta un interprete de python nuevo lo que hace que se ejecute el código de nuevo entero y solo se le pasan los recursos necesarios). Esto se debe a que en Windows no esta disponible el fork, que si esta disponible en Unix.

Para paliar este problema lo máximo posible se han ido haciendo comprobaciones de PIDs a lo largo del código para delimitar a que zonas puede acceder cada interprete de python de forma que los procesos hijos no ejecuten la totalidad del código y solo carguen las redes neuronales y ejecuten las predicciones.El hecho de que se use spawn en vez de fork hace que el algoritmo sea más lento lo que no sería el funcionamiento óptimo de este, por lo tanto para un trabajo futuro se debería migrar a Unix para poder eliminar este factor.

Arquitectura global

Esta arquitectura es más simple que la anterior, en si esta compuesta solo por 1 modelo idéntico a los comentados en los apartados anteriores pero este ha sido entrenado para detectar todos los tipos de tráfico a la vez.

Para el desarrollo de esta, solo se ha entrenado un modelo con un conjunto de datos que consistía en la unión de los datasets con los principales tipos de tráfico. Las ventajas de esta arquitectura son su simpleza y su velocidad de ejecución, pero su principal desventaja es que no cuenta con el nivel de precisión que tiene la arquitectura en paralelo ya que cuenta con 1 sola red y no con 3.

La comparativa de estas 2 arquitecturas se hará en el siguiente capítulo.

3.4.4. Eliminación de la red LSTM

Después de haber diseñado las aproximaciones a los IDS, se creyó oportuno replantear el modelo propuesto por Vinayakumar para estudiar si era posible seguir obteniendo resultados tan satisfactorios con un modelo más rápido y menos costoso computacionalmente.

A la hora de pensar como cumplir con los objetivos marcados anteriormente, la modificación más factible para este fin es minimizar el número de capas de la red y así tener un procesamiento más veloz. La elección de la parte a eliminar ha sido relativamente sencilla ya que el modelo tiene 2 elementos claramente diferenciados, una red CNN y una red LSTM. El elemento eliminado ha sido la red LSTM.

Esta eliminación esta además respaldada por la propia investigación de Vinayakumar ya que en ella se evalúan diferentes arquitecturas y se puede comprobar que las arquitecturas basadas únicamente en redes CNN tienen un desempeño altamente satisfactorio.

Como contraparte de esta eliminación, es esperable que empeoren los resultados, pero esto solo se puede comprobar después de hacer las pruebas y evaluar las salidas. La situación ideal se dará si se consiguen mantener los resultados o estos solo empeoran ligeramente.

3.5. Conclusiones

A lo largo de este capítulo se han comentado una serie de desarrollos llevados a cabo en el proceso de diseño del modelo y corrección de errores encontrados en este proceso. Esta etapa es la más relevante de todo el trabajo, por lo tanto es la etapa en la que más tiempo se ha invertido ya que era necesario un buen planteamiento de esta para la obtención de resultados satisfactorios. Su desarrollo ha sido paralelo al de la etapa de pruebas y resultados ya que para la toma de decisiones ha sido necesario basarse en resultados.

Hacia el final de este apartado se han comentado desarrollos clave y conceptos clave para una implementación final y fundamentales para el crecimiento del sector. Son necesarias más investigaciones de este tipo y de mayor profundidad en estos conceptos mencionados para que algún día llegue a ser una tecnología de grandes beneficios.

4

Pruebas y Resultados

4.1. Introducción

La finalidad de este capitulo es mostrar las pruebas realizadas a lo largo del trabajo para la evaluación tanto de los datos como de los modelos planteados en el apartado anterior, además, se comentarán los principales resultados obtenidos y sus implicaciones en la toma de decisiones propia de las diferentes modificaciones o correcciones aplicadas a el procesamiento de los datos, el modelo o la evaluación de estos resultados.

Este es el capitulo donde se mostrará si la investigación y desarrollo llevado a cabo en este trabajo ha tenido como resultado un modelo eficaz, veraz y aplicable. También se comentará un estudio más profundo de la investigación realizada por Vinayakumar R. y sus fortalezas y defectos.

4.2. Descripción de las pruebas realizadas

Durante la etapa de realización del trabajo se han hecho diferentes pruebas tanto a los datos de entrada como a los modelos, al depender el desarrollo cronológico del proyecto de estas pruebas, estas también se comentarán siguiendo la linea temporal del desarrollo.

Las pruebas realizadas sobre los conjuntos de datos han estado enfocadas a la comprensión de estos y de sus parámetros para así poder comprender mejor los resultados finales obtenidos. Estas pruebas se podrían dividir en dos tipos según su finalidad.

1.– Corrección y comprensión de conjuntos.

2.– Modificaciones de conjuntos para pruebas del modelo.

Las pruebas de corrección y comprensión de los conjuntos han sido las más simples de todas, ya que han consistido en evaluar los conjuntos parámetro a parámetro y determinar posibles errores en estos que pudieran llegar a producir un funcionamiento erróneo del modelo.

4.2.1. Replicación de la investigación de Vinayakumar R.

Pruebas sobre dataset

Sobre el primer conjunto de datos KDD-CUP 99', la comprensión se ha llevado a cabo utilizando diferentes instrucciones para acceder al contenido de cada fila o columna y así poder comprender valores típicos de cada parámetro y posibles desviaciones de estos. Estas pruebas han arrojado varios factores a tener en cuenta no comentados por Vinayakumar.

El primer factor preocupante encontrado en este dataset es la relación entre tráfico maligno y benigno, no es realista ya que el tráfico benigno representa el 19.859 % del conjunto y los ataques el 80.141 %. Esto hace que la relación entre ambos sea que el tráfico maligno es aproximadamente 4 veces mayor que el benigno. Este factor es extremadamente relevante ya que hace que este dataset no se pueda considerar de ninguna manera aplicable al desarrollo de un sistema realista y la única utilidad que tiene es en investigaciones o competiciones sin intención de diseño de un sistema aplicable.

Si ahondamos más en lo comentado anteriormente y revisamos los datos de tráfico por tipo (en python esto se puede hacer con un simple .value_counts() de la columna 41 del dataset) obtendremos los datos mostrados en E.1. Solamente viendo esta tabla ya se puede observar que dentro del tráfico maligno hay un desbalanceo extremadamente preocupante, para una visualización más clara de este problema se diseña la siguiente tabla:

Tipo de tráfico	Muestras totales	%	Tipo de tráfico	Muestras totales	%
DoS	3.883.370	79.278 %	**Normal**	972.781	19.859 %
Probe	41.102	0.839 %	**R2L**	1126	0.023 %
U2R	52	0.001 %			

Tabla 4.1: Porcentajes de KDD-CUP 99'

Queda claro después de obtener esta tabla que este dataset tiene gran cantidad de fallos en la repartición de las clases. Esto sería admisible si se debiese al hecho de que el dataset intenta replicar tráfico real, pero al no ser así, tener un dataset con una clase tan predominante y las demás con porcentajes tan bajos es casi como tener un dataset que solo cuente con las clases DoS y normal.

Estos problemas encontrados en el dataset al revisarlo no se deben arreglar, ya que como la finalidad de esta parte del proyecto es replicar el trabajo de Vinayakumar, no se deben alterar los datos para usar los mismos que el.

La principal función de estos problemas será la comprensión de los resultados obtenidos en el siguiente apartado, ya que serán muy influyentes en los porcentajes de eficacia de la red para cada clase.

En cuanto a correcciones realizadas en el conjunto, solo se han realizado las que se han comentado

con anterioridad en estas memoria, era necesario hacer una transformación a valores numéricos de parámetros no numéricos y se ha hecho.

Pruebas sobre modelo

Las pruebas realizadas sobre este modelo han estado enfocadas a replicar parte del desarrollo de la investigación de Vinayakumar. En concreto se ha intentado replicar su predicción multiclase para el modelo de CNN+LSTM.Como se ha comentado con anterioridad, se ha usado este modelo y no otro porque es general es de los modelos que mejores resultados obtenía y se prestaba a futuras modificaciones.

Para evaluar si la replicación ha sido satisfactoria o no se han hecho pruebas para obtener algunos de los parámetros que el muestra y otros determinados por el alumno, como son:

- Entrenamiento y test del modelo para evaluar exactitud.
- Obtención de micro y macro avg de determinados parámetros.
- Obtención de matrices de confusión.

La primera prueba comentada y llevada a cabo es la de el test del modelo para evaluar su accuracy. Para poder hacer estas pruebas antes del entrenamiento del modelo hay que configurar determinados parámetros en la red a través de las instrucciones mostradas en L.3.

Este entrenamiento del modelo corresponde a la primera prueba ya que nos da una idea de la calidad del modelo que se esta desarrollando, aunque el parámetro accuracy no sea el más indicado para medir un dataset con tanto desbalanceo debido al acuraccy paradox (A.1).

Para hacer una prueba más específica y útil, usamos los datos de test que se han generado con anterioridad. Para ejecutar esta prueba debemos usar el código en L.4.

Con esto se puede hacer una sencilla evaluación del modelo entrenado, pero esta evaluación solo arrojaría datos globales de accuracy, cosa que no es lo que se busca ya que este parámetro es poco fiable.

La siguiente prueba que se ha realizado sobre esta replicación, es una continuación de las realizadas por Vinayakumar. En su investigación solo calcula otros parámetros relevantes como F1-Score, Precision y Recall cuando esta ejecutando predicción binaria, pero gracias al cálculo de medias como micro average y macro average, somos capaces de aunar las contribuciones de cada tipo de trafico para obtener unos valores globales de estos parámetros para clasificación multiclase. Las nuevas medidas comentadas se desarrollan en el apéndice F.

Por último, se obtiene la matriz de confusión del modelo para evaluar el efecto del desbalanceo de los datos y así tener una visión realista de el funcionamiento del modelo y poder sacar conclusiones de la investigación de Vinayakumar.

4.2.2. Desarrollo sobre CSE-CIC-IDS2018

Para el desarrollo de esta parte del trabajo también se han ido haciendo pruebas intermedias para ir evaluando los resultados y detectar posibles errores antes de obtener unos resultados definitivos del modelo. Este apartado comprende tanto a las pruebas realizadas para evaluar el modelo sobre el nuevo conjunto de datos como a las pruebas realizadas el evaluar la eliminación de la red LSTM.

Pruebas sobre dataset

Parte de las pruebas realizadas para la comprensión de este dataset ya se han comentado en el apartado 3.4.1. Los factores más preocupantes de este conjunto de datos son similares a los del dataset anterior aunque este nuevo conjunto no cuenta con todos ellos y los que tiene pueden llegar a estar justificados.

En cuanto al reparto de clases del dataset, aunque también sufre de desbalanceo, este es más realista ya que la clase predominante es la de tráfico normal. Aun así, hay que tener en cuenta que el factor de relación entre tráfico benigno y maligno es de 4.9 a favor del tráfico benigno.

Al hacer un estudio más detallado de los diferentes parámetros y sus valores para todos los subconjuntos de datos de este dataset, haciendo diferentes pruebas para ir evaluando poco a poco todas las columnas y filas, se han detectado aparte varios problemas serios que no se vieron en el dataset anterior.

Estos problemas están relacionados con un mal tratamiento de los datos a la hora de añadirlos a los conjuntos. El primer problema que nos podemos encontrar es que haya filas enteras en diferentes conjuntos que no tengan valores asociados si no que simplemente repliquen los nombres de cada parámetro, lo que convierte a toda la fila en un potencial problema que generaría un error a la hora de entrenar el modelo. Este problema tiene fácil solución y es eliminar por completo la fila, esto se hace automáticamente todas las veces que sea necesario hasta eliminar todas las filas erróneas.

Al revisar los parámetros uno por uno haciendo uso de las funciones proporcionadas por la librería Pandas [35] para tratamiento de datos, también se ha localizado otro error debido a operaciones erróneas de CICFlowmeter y la falta de revisión por parte de los creadores del dataset. Estos errores generan que haya flujos que cuentes con valores no validos en alguno de sus parámetros como son $\pm NaN$ o $\pm \infty$. Esto también es sencillo de corregir usando la instrucción *replace* de pandas.

Gracias a las pruebas realizadas ha sido posible prevenir posibles errores en la clasificación debidos a datos incorrectos. También se ha conseguido entender los diferentes parámetros y la forma en la que podrían llegar a influir en la toma de decisiones de la red.

Pruebas sobre el modelo

Las pruebas realizadas sobre este modelo se pueden considerar una evaluación más exhaustiva del modelo que las que lleva a cabo Vinayakumar, ya que la finalidad de estas es profundizar en el funcionamiento de este modelo para entender sus puntos fuertes y débiles, además de entender las posibles aplicaciones de un modelo de estas características.

Las pruebas consisten en la extracción del valor de accuracy (comentando anteriormente), extracción de parámetros como F1-Score, Recall y Precision para cada clase y en conjunto y extracción de matrices de confusión.

Algunas de las pruebas se han comentado ligeramente al explicar las pruebas realizadas sobre el modelo aplicado sobre KDD-CUP 99', ahora se detallarán su finalidad y su procedimiento ya que es mas interesante que estas sean comentadas ahora para la evaluación de este nuevo desarrollo.

La extracción de los parámetros comentados anteriormente se hace de manera automática mediante la función *classification_report* de scikit learn que nos permite conocer estos valores para cada uno de las clases que se encuentran en los datos de test.

Es de vital importancia conocer el valor de estos parámetros ya que gracias a ellos podemos comprender aspectos del modelo que solo con accuracy no veríamos. La explicación y finalidad de cada uno de estos parametros se comenta en el apéndice F.

Estos parámetros también están estrechamente ligados con las diferentes matrices de confusión por lo que también se han extraído.

A partir de las pruebas realizadas para extraer estos valores y del estudio de estas se ha ido modificando tanto el modelo como los datos de entrada de las maneras que ya se han comentado, eliminando elementos del conjunto de datos y la red LSTM del modelo. Estas pruebas también han servido para evaluar la idea de que este modelo sea aplicable a un IDS o al menos que siga siendo interesante el estudio de los posibilidades.

4.3. Resultados obtenidos

Una vez comentados los diferentes modelos a los que se les han aplicado las pruebas, el como y el porqué de estas, ahora se comentarán los diferentes resultados obtenidos en cada uno de los modelos. Se divide la presentación de estos resultados siguiendo el mismo esquema que se ha usado a lo largo de toda la memoria.

4.3.1. Replicación de la investigación de Vinayakumar R.

En este primer apartado se verán los resultados obtenidos del intento de replicación y la evaluación posterior a la replicación.

Vinayakumar entrena el modelo durante 500 epochs lo que hace que su modelo sufra de mucho overfitting. Para comprobar esto se ha entrenado el modelo durante 240 epochs, la diferencia de epochs es debida a la diferencia de hardware usado.

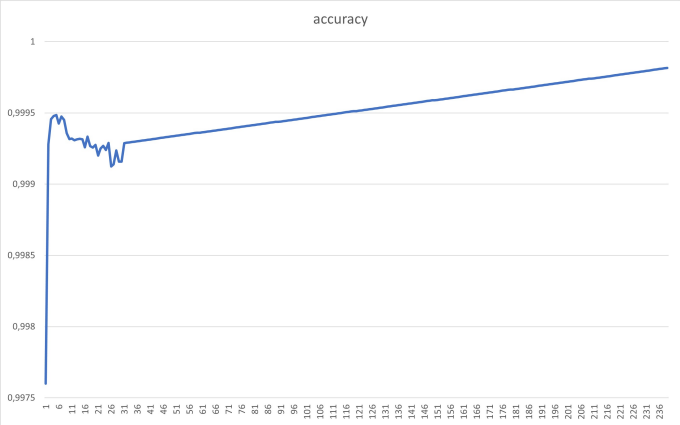

Figura 4.1: Estudio del overfitting en modelo de Vinayakumar R.

Se puede ver en la imagen que el modelo acaba sufriendo overfitting. En este trabajo se ha preferido usar solo 10 epochs ya que según la gráfica mostrada, parece un numero de epochs con el que es muy posible que nos acerquemos a los valores mostrados en su investigación.

Utilizando 10 epochs el mejor valor de accuracy se obtiene en la iteración 5 y con esos pesos para el modelo se obtienen los siguientes resultados:

$$Accuracy = 99{,}952\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	58.96 %	53.209 %	54.124 %	1.224.608
Weighted average	99.95 %	99.951 %	99.949 %	1.224.608

Tabla 4.2: Parámetros globales de replicación

Como se ve, se ha conseguido obtener una exactitud/accuracy ligeramente superior a la obtenida por Vinayakumar R., esto posiblemente sea debido a diferencias en el pre-procesamiento y el overfitting que sufre su modelo.

Respecto a los parámetros globlales obtenidos, tiene más sentido evaluarlos a la vez que se evalúan los resultados obtenidos por clase, tanto medidas como matrices de confusión. Por falta de espacio estos elementos se pondrán en el anexo G.

Observando tanto la tabla como la matriz de confusión se puede asegurar lo que ya se suponía durante las pruebas realizadas al dataset, el valor de accuracy es un parámetro poco realista que no refleja la realidad de la aplicabilidad del modelo.

El funcionamiento del modelo se observa con mayor claridad en la matriz de confusión de este, en esta se pueden observar claramente varias situaciones que luego se pueden confirmar viendo los datos de las clases afectadas. En esta matriz se pueden distinguir las predicciones en 2 grupos de clases claros, las correctamente detectadas y las que se confunden prácticamente en su totalidad con trafico benigno. Si después de revisar la matriz revisamos la tabla del mismo anexo, veremos que prácticamente la totalidad de las clases mal clasificadas tienen parámetros que son 0 o son muy próximos a este. Por el otro lado, tenemos un grupo de clases muy bien detectadas y con unos parámetros por encima de 80 %.

Estas dos situaciones están altamente relacionadas ya que ambas se deben al support (número de muestras de la clase) que tienen asociado, este parámetro mantiene el mismo porcentaje en la etapa de entrenamiento lo que hace que el modelo tenga mejores métricas de detección para las clases predominantes que generalmente suelen ser las DoS y DDoS.

En cuanto a predicción por tipos de ataque, el modelo detecta mejor los tipos DoS, DDoS y Probe, aunque para algunos ataques R2L como warezclient y guess_password también tiene buenos registros.

Como último detalle a comentar esta el hecho de que prácticamente no hay confusión del modelo entre clases de ataques, si se clasifica mal un ataque se suele confundir con tráfico normal pero no con otro tipo de ataque.

4.3.2. Evaluación del modelo sobre CSE-CIC-IDS2018

En este segundo apartado se verán los resultados obtenidos del desarrollo del modelo para su utilización sobre el dataset CSE-CIC-IDS2018. Primero se comentará el estudio sobre la versión con Timestamp y posteriormente el estudio de la versión sin Timestamp.

Para optimizar el espacio y el número de muestras se mostrarán los resultados de las detecciones aplicadas a los diferentes tipos de ataque (DoS, fuerza bruta e infiltración). Se debe tener en cuenta que los datos que se van a mostrar son extraídos de conjuntos de datos generados con todos los subconjuntos de datos que contienen ese tipo de ataque en concreto. Esta aproximación nos permitirá además observar con mayor claridad los puntos fuertes y débiles de detección de esta tecnología en tiempo real.

Siguiendo el mismo razonamiento expuesto en el apartado anterior para las epoch, las siguientes muestras se han extraído usando 15 epochs para reducir el overfitting.

Datos de entrada con Timestamp

Para esta primera versión tenemos los siguientes parámetros globales:

Fuerza Bruta

$$Accuracy = 99{,}96\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	79.24 %	65.08 %	69.89 %	524.288
Weighted average	99.957 %	99.96 %	99.96 %	524.288

Tabla 4.3: Parámetros globales de ataques de Fuerza bruta en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo H.1.

Infiltración

$$Accuracy = 86{,}21\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	82.04 %	63.36 %	66.89 %	236.043
Weighted average	85.17 %	86.21 %	85.68 %	236.043

Tabla 4.4: Parámetros globales de ataques de Infiltración en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo H.2.

DoS/DDoS

$$Accuracy = 99{,}85\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	98.76 %	99.3 %	99.02 %	786.431
Weighted average	99.86 %	99.85 %	99.85 %	786.431

Tabla 4.5: Parámetros globales de ataques de DoS y DDoS en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo H.3.

Una vez se tienen todos los parámetros disponibles, es bastante sencillo entender el comportamiento del modelo. Este modelo tiene unos indicadores extremadamente altos para el trafico DoS o DDoS, detecta de manera muy satisfactoria estas clases. La matriz de confusión nos confirma la eficacia de este modelo para la detección de este tipo de ataques, lo que nos indica que este modelo

es muy viable para la implementación en un IDS para detección de trafico DoS/DDoS. Este hecho se puede deber a la similitud de los diferentes flujos de cada clase o la gran cantidad de muestras.

Respecto a los otros 2 tipos de ataques, se puede observar que el tráfico de fuerza bruta también es detectado de manera satisfactoria, pero con una diferencia clara respecto del tipo anterior. Esta diferencia es el error cometido en la clasificación de determinados ataques posiblemente debido a la falta de muestras en los datos de train.

El tráfico de infiltración es sin duda el peor detectado por este modelo, si se revisan los indicadores por clase, se observa que para la clase de Infiltración tenemos un Recall extremadamente bajo. Esto nos quiere decir que el modelo no es capaz de reconocer patrones en esta clase debido a la diferencia entre estos flujos. Lo que hace que un IDS basado en este modelo no sea recomendable para la detección de tráfico de infiltración.

Un aspecto positivo del modelo, es que en los casos en los que clasifica mal un flujo, prácticamente todos estos tráficos mal clasificados suelen detectarse como trafico benigno. Esto es positivo porque nos indica que el modelo es capaz de diferenciar todos los ataques entre si.

Datos de entrada sin Timestamp

Si ahora se elimina el parámetro Timestamp obtenemos los siguientes resultados:

Fuerza Bruta

$$Accuracy = 99{,}971\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	81.97 %	63.88 %	69.24 %	524.288
Weighted average	99.968 %	99.97 %	99.97 %	524.288

Tabla 4.6: Parámetros globales de ataques de Fuerza bruta en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo I.1.

Infiltración

$$Accuracy = 82{,}746\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	41.37 %	50 %	45.28 %	236.043
Weighted average	68.47 %	82.75 %	74.93 %	236.043

Tabla 4.7: Parámetros globales de ataques de Infiltración en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo I.2.

DoS/DDoS

$$Accuracy = 99{,}861\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	98.74 %	98.88 %	98.81 %	786.431
Weighted average	99.86 %	99.86 %	99.86 %	786.431

Tabla 4.8: Parámetros globales de ataques de DoS y DDoS en CSE-CIC-IDS2018

Se pueden encontrar los indicadores por clase en el anexo I.3.

Si se revisa ataque por ataque, se observa que generalmente los indicadores han mejorado o empeorado ligeramente respecto del modelo sobre datos sin Timestamp, esto no aplica a los ataques de infiltración, ya que sus mediciones actuales son mucho peores.

Esto hace que la teoría de que determinados ataques como los de infiltración eran detectados debido a criterios basados en la hora exacta de estos cobre fuerza. En un IDS esta es la situación que no quieres tener ya que quieres que este sea capaz de detectar tráfico basándose en características de flujos/paquetes y del trafico, no específicamente en la hora a la que se recibe ese trafico.

Considerando ahora los ataques de fuerza bruta, se observa que los patrones que identifica la red han mejorado y por eso aumentan sus mediciones generales.Para el caso de los ataques DoS/DDoS, su detección ha empeorado ligeramente, pero para nada un porcentaje preocupante. Se puede asumir que el nivel de detección se ha mantenido. Esto sumado al hecho de que se ha eliminado la dependencia horaria de los paquetes, hace que este sea un modelo mucho más interesante de aplicar.

Comparativa con\sin Timestamp

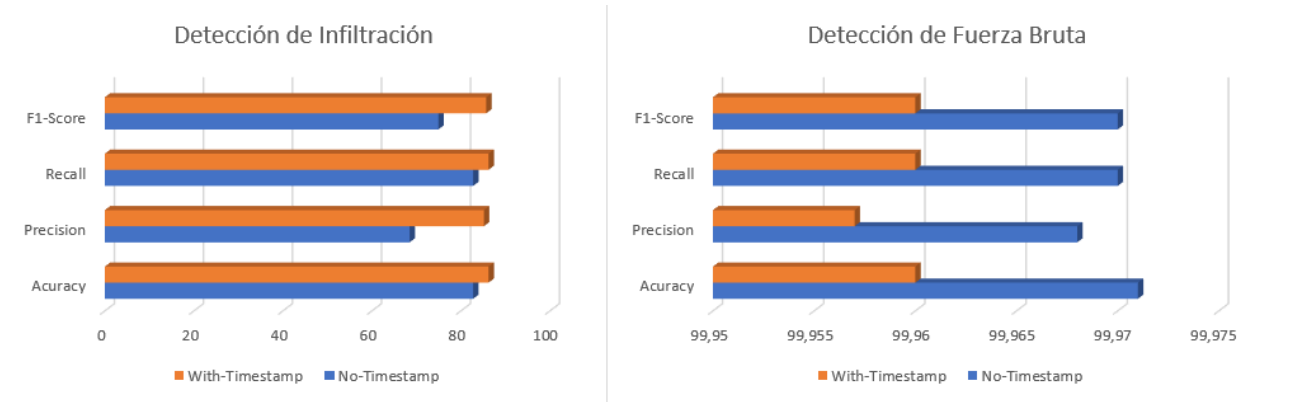

Figura 4.2: Comparativa de modelos para Infiltración y Fuerza Bruta

Para esta comparativa se usan las micro medias debido a que son recomendables para datasets desbalanceados. En estas tres gráficas se comparan los parámetros globales del sistema.

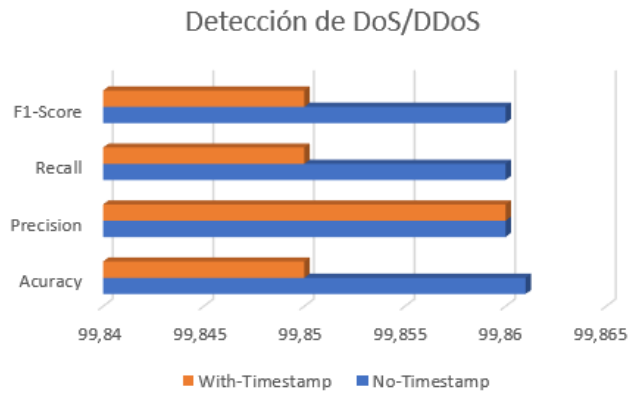

Figura 4.3: Comparativa de modelos para DoS/DDoS

Estas comparaciones sirven para comprender visualmente las diferencias entre los dos modelos. Como se ha comentado, los datos de infiltración son malos en ambos casos. Esto no se debe a ningún error en el modelo si no que proviene de la diferencia entre flujos de infiltración. Si se evalúan los resultados más a fondo para ambos modelos, se observa el mismo defecto en ambos. Las detecciones de tráfico maligno son pésimas y los indicadores superiores al 80 % solo se consiguen debido a una buena clasificación del tráfico benigno que representa la mayor parte de los datos.

Para clases de fuerza bruta observamos una mejora del 0.02 % en todos los indicadores al eliminar el Timestamp. La razón es que, al eliminar este parámetro, el modelo ha reconocido un patrón diferente lo que ha aumentado su eficacia. En ambos modelos obtenemos una precisión ligeramente inferior al resto de indicadores. Esto se debe a que en ambos modelos se confunden clases a la hora de clasificar. Viendo la matriz de confusión de ambos modelos se observa este suceso: Muchos de los errores de clasificación vienen de confundir la clase con BruteForce-Web, esto muy posiblemente se deba a que al tener poco support para esta clase, no la ha aprendido correctamente.

Para las clases de DoS y DDoS, la eliminación del Timestamp supone una mejora de 0.01 % en todos los indicadores salvo precisión, que mantiene un valor extremadamente alto debido a que en este modelo practicamente no hay confusión de clases.

4.3.3. Desarrollo de IDS

En este apartado se mostrarán los resultados obtenidos para las dos aproximaciones a IDS comentadas.

Arquitectura con red global

En esta arquitectura se detectan todos los ataques en la misma red, por lo tanto esta tiene que clasificar muchas más clases. Si corremos sobre este modelo las mismas pruebas que sobre los anteriores

obtenemos:

$$Accuracy = 95{,}728\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	67.296 %	64.07 %	63.59 %	1.546.762
Weighted average	94.49 %	95.73 %	94.386 %	1.546.762

Tabla 4.9: Parámetros globales de arquitectura global

Se pueden encontrar los indicadores por clase en el anexo J.

Como era de esperar, la detección de las diferentes clases a empeorado al aplicar sobre el mismo modelo unos datos menos específicos pero aún así se mantiene en unos niveles aceptables. Los valores de Weighted average siguen siendo altos, pero esto es debido a la gran cantidad de flujos de tráfico de infiltración que hay en el conjunto de datos y que se clasifican correctamente.

Si revisamos la matriz de confusión y los parámetros de clases, se observa que todo lo comentado anteriormente para detecciones de cada tipo se puede aplicar a este modelo también, se detectan mejor los ataques DDoS/DoS y la segunda mejor detección aplica a la clase Fuerza Bruta. El tipo Infiltración se detecta igual de mal.

Un aspecto negativo a destacar de este modelo es que si que existen mal interpretaciones de clases que se detectan como otras clases.

Arquitectura con redes en paralelo

Como para esta arquitectura no se han modificado las redes especificas de cada clase de ataque, los resultados mostrados en 4.3.2 siguen siendo los aplicables a esta arquitectura.

El único factor a tener en cuenta comentado con anterioridad en el trabajo, es que al usar 3 redes neuronales diferentes, es posible que varias detecten un ataque en el mismo flujo, esto no afectaría a un teórico IDS ya que en este caso alertaría de la posibilidad de esos varios ataques y se tomarían medidas respecto de todos. Otra opción para un posible IDS sería en estos casos usar los indicadores de calidad de detección de cada clase comentados en el apéndice I para tomar una decisión para asumir un ataque u otro.

Comparativa

Para comparar los resultados obtenidos hay que basarse en en dos aspectos, la “calidad"de las detecciones y el tiempo en llevarlas a cabo. Este último aspecto no se ha calculado en ninguna arquitectura debido al hecho de que no se han terminado de implementar con tráfico en tiempo real, pero

se puede inferir de su estructura.

En cuanto a calidad de las detecciones, es evidente que la arquitectura con redes en paralelo ofrece una medición mucho mejor debido a la especialización de cada una de ellas.

Respecto al tiempo necesario para procesar los datos de entrada, la arquitectura con la red global es ligeramente más rápida debido al hecho de que la arquitectura en paralelo tiene instrucciones extra para generar el paralelismo y unir los resultados en el sistema de votación. Esta diferencia de tiempo se puede paliar con hardware de gran potencia computacional.

Debido a esto último comentado, la recomendación del alumno es aplicar el esquema con redes en paralelo.

4.3.4. Estudio de la posibilidad de eliminar la red LSTM del sistema

Este último apartado servirá para evaluar un IDS que solo cuente con una red CNN. Como después de evaluar las dos arquitecturas anteriores se ha decidido usar la arquitectura con redes en paralelo, siguiendo esta idea, se entrenan 3 modelos por separado y se obtienen los siguientes resultados:

Fuerza Bruta

$$Accuracy = 99{,}962\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	66.62 %	63.88 %	65.12 %	524.288
Weighted average	99.95 %	99.96 %	99.95 %	524.288

Tabla 4.10: Parámetros globales de ataques de Fuerza bruta sin LSTM

Se pueden encontrar los indicadores por clase en el anexo K.1.

Infiltración

$$Accuracy = 82{,}737\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	66.09 %	50.63 %	46.83 %	236.043
Weighted average	77.11 %	82.74 %	75.45 %	236.043

Tabla 4.11: Parámetros globales de ataques de Infiltración sin LSTM

Se pueden encontrar los indicadores por clase en el anexo K.2.

DoS/DDoS

$$Accuracy = 99{,}899\,\%$$

	Precisión	Recall	F1-Score	support
Macro average	98.70 %	99.62 %	99.15 %	786.431
Weighted average	99.90 %	99.89 %	99.90 %	786.431

Tabla 4.12: Parámetros globales de ataques DDoS/DDoS sin LSTM

Se pueden encontrar los indicadores por clase en el anexo K.3.

La situación ideal sería que los indicadores no se vieran afectados o que se vieran ligeramente afectados. El primer tipo de ataque que se comentará será el de infiltración, es el que peores medidas ofrece y es el menos relevante. Estas medidas no se ven demasiado alteradas, pero aun así mejoran, lo que es una muy buena señal de cara a comprobar los demás ataques. También es una reafirmación de que esta arquitectura no es un buen detector de este tipo de ataques, claramente observabable con el parametro Recall ya que este tiene un valor de 1.6 % para la clase.

En cuanto a los otros dos tipos, si comparamos los resultados obtenidos con y sin LSTM para los ataques del tipo DoS/DDoS, observaremos que las mediciones se han mantenido muy similares, mejorando la detección de varias clases. En cuanto a los ataques de fuerza Bruta, el comentario es idéntico.

4.3.5. Conclusión

En este capítulo se han mostrado todas las pruebas realizadas a los diferentes modelos a la hora de evaluar su rendimiento. Como se ha comentado con anterioridad, estas pruebas también tienen como finalidad el detectar posibles mejoras o fallos y han sido muy útiles para ese fin.

Los resultados finales obtenidos en la parte de desarrollo han resultado ser muy positivos, obteniendo valores de clasificación muy altos en cada uno de los modelos, aunque hay que tener en cuenta que estos ataques fueron registrados hace 2 años y es muy posible que los actuales sean diferentes. Aun así, tanto el modelo CNN+LSTM y el modelo CNN parecen ser muy buenos detectores de tráfico. Otro punto positivo es que ningún modelo parece sufrir en exceso de overfitting.

Como contraposición se ha descubierto que la investigación de Vinayakumar tenia algunos parámetros muy positivos (los mostrados por el), pero luego varias clases sufrían de mala predicción y el dataset utilizado parece ser poco óptimo para el estudio de un IDS.

5

Conclusiones y trabajos futuros

5.1. Conclusiones

Para la realización de este trabajo de Fin de Carrera, se han ido dando diferentes pasos como son una primera primera parte de análisis, selección, replicación, evaluación, desarrollo y optimización con la finalidad de presentar varias aproximaciones posibles para un IDS.

Los objetivos marcados para este trabajo se han cumplido y extendido, ya que ha dado tiempo para hacer una investigación más exhaustiva de la que se planteó en un primer momento.

Tras realizar un análisis del estado del arte y así poder tener una primera impresión de la tecnología usada, algoritmos planteados y momento actual del desarrollo del campo, se ha conseguido comprender los conceptos más relevantes para el desarrollo de este trabajo y para el desarrollo del sector. Debido a estos conocimientos adquiridos, se han identificado varios problemas clave como pueden ser la antigüedad de las principales investigaciones y el uso de métricas no del todo exactas. Esto ha dado pie al desarrollo de este trabajo para intentar modernizar una aproximación en concreto que parecía ser muy prometedora.

Durante la fase de diseño y desarrollo, siguiendo el flujo antes comentado, el primer trabajo realizado ha sido la replicación de una investigación prometedora para su evaluación. Tanto para este trabajo como para el resto los conocimientos adquiridos en la etapa de investigación han sido extremadamente útiles ya que han permitido una mejor verificación de este modelo a la vez que encontrar debilidades no mencionadas como la pobre clasificación de determinados ataques. El segundo trabajo realizado ha sido el desarrollo de un modelo más moderno capaz de detectar eficazmente gran cantidad de amenazas. Para el desarrollo de este trabajo se han ido haciendo diferentes aproximaciones y evaluando los aspectos más positivos y negativos de cada una de ellas para así poder tomar decisiones basadas en métricas y modificar el modelo para su optimización. En la siguiente figura se detalla cual ha sido el mejor modelo desarrollado para cada tipo de ataque.

	DoS/DDoS	Fuerza Bruta	Infiltración
Mejor modelo	Sin Timestamp y sin LSTM	Sin Timestamp	Con Timestamp
Accuracy	99.899%	99,971%	86,21%
Precisión	99,90%	99,968%	85,17%
Recall	99,89%	99,97%	86,21%
F1 Score	99,9%	99,97%	83,45%

Figura 5.1: Mejor modelo para cada tipo

Como resultado final, se plantean dos modelos optimizados usando uno una arquitectura CNN y el otro una arquitectura CNN+LSTM. Estos dos modelos aplicados a tráfico quasi-real obtienen unos valores de acierto superiores al 99.95 % en la detección de los dos tipos principales de ataque pero se muestran ineficaces en la detección de ataques de Infiltración.

En conclusión, para este trabajo han sido necesarios conocimientos de campos como TSM, Redes I y Redes II para la comprensión de aspectos esenciales del tráfico de red y el uso y comprensión de redes neuronales. Este trabajo aúna conocimientos tanto de la rama de telemática como de la rama de imagen y sonido y en un futuro se plantea relacionarlo con conocimientos de la rama de electrónica adquiridos en asignaturas como APS.

Todos los códigos usados en este TFG se pueden consultar en el GitHub del estudiante(ID: javieradelgado) , en el repositorio *AnalisisDeTraficoCNN_LSTM* [36].

5.2. Trabajo Futuro

Este proyecto aún tiene mucho campo de mejora y desarrollo, a continuación se comentarán algunas de estos posibles desarrollos:

- **Implantación del algoritmo en entorno en tiempo real**: Este desarrollo no es sencillo y permitirá usar el modelo en redes reales para monitorizar el tráfico. Para este fin debe de modificarse el método de entrada de datos y el tratamiento de estos antes de pasarse por la red. También debe modificarse la salida para que esta sea leída y se tomen decisiones automáticamente al obtenerla
- **Optimización de los recursos usados**: Tensorflow funciona por defecto sobre SSE 4.1 en cpu, para mejorar las prestaciones del sistema se puede modificar la ISA que use la cpu o migrar el modelo a elementos más especializados como son los ASICS

- **Modificación de la arquitectura para aumentar la detección de infiltración**: Este desarrollo consistiría en volver a hacer una investigación de diferentes modelos de redes neuronales y sus eficiencias en la detección de trafico de Infiltración para así poder sustituir el modelo poco eficiente de la arquitectura en paralelo por otro modelo con mejores resultados y así aumentar la eficiencia global del sistema
- **Añadir método para que sistema se entrene a la vez que detecte**: Este desarrollo mejoraría en gran medida la capacidad de detección de la arquitectura además de hacer que esta pudiese "mantenerse al día". Mezclando tanto aprendizaje supervisado como no supervisado se podría generar un modelo de grandes capacidades

Bibliografía

[1] D. Faggella, "What is machine learning," 2020. Disponible en https://emerj.com/ai-glossary-terms/what-is-machine-learning/ (Accedido el 11/06/2021).

[2] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer, 2018.

[3] M. K. Pratt, "What is an intrusion detection system? how an ids spots threats," 2018. Disponible en https://www.csoonline.com/article/3255632/what-is-an-intrusion-detection-system-how-an-ids-spots-threats.html (Accedido el 14/08/2020).

[4] M. Azizjon, A. Jumabek, and W. Kim, "1d cnn based network intrusion detection with normalization on imbalanced data," in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 218–224, 2020.

[5] UNSW, "Unswnb15 dataset," 2015. Disponible en https://research.unsw.edu.au/projects/unsw-nb15-dataset (Accedido el 20/08/2020).

[6] A. R. Shaaban, E. Abd-Elwanis, and M. Hussein, "Ddos attack detection and classification via convolutional neural network (cnn)," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 233–238, 2019.

[7] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1222–1228, 2017.

[8] UCI, "University of california, irvine." Disponible en https://uci.edu/ (Accedido el 15/09/2020).

[9] MIT, "Darpa 1998 dataset," 1998. Disponible en https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset (Accedido el 22/09/2020).

[10] UCI, "The fifth international knowledge discovery and data mining tools competition," 1999. Disponible en http://kdd.ics.uci.edu/databases/kddcup99/task.html (Accedido el 22/09/2020).

[11] UCI, "Características de kddcup 99'," 1999. Disponible en http://kdd.ics.uci.edu/databases/kddcup99/kddcup.names (Accedido el 22/09/2020).

[12] CSE and CIC, "Dataset cse-cic-ids2018," 2018. Disponible en https://www.unb.ca/cic/datasets/ids-2018.html (Accedido el 7/01/2021).

[13] I. Sharafaldin., A. Habibi Lashkari., and A. A. Ghorbani., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, pp. 108–116, INSTICC, SciTePress, 2018.

[14] Jupyter, "Jupyter notebook." Disponible en https://jupyter.org/ (Accedido el 25/09/2020).

[15] Anaconda, "Distribución anaconda." Disponible en https://www.anaconda.com/ (Accedido el 25/09/2020).

[16] TensorFlow, “Biblioteca tensorflow.” Disponible en https://www.tensorflow.org/?hl=es-419 (Accedido el 25/09/2020).

[17] AMD, “Rocm,” 2018. Disponible en https://rocmdocs.amd.com/en/latest/index.html (Accedido el 30/09/2020).

[18] Keras, “Biblioteca keras.” Disponible en https://keras.io/ (Accedido el 25/09/2020).

[19] CIC and A. H. Lashkari, “Documentación de cicflowmeter,” 2018. Disponible en https://www.unb.ca/cic/research/applications.html (Accedido el 20/02/2021).

[20] CIC, “Canadian institute forcybersecurity.” Disponible en https://www.unb.ca/cic/ (Accedido el 07/01/2021).

[21] A. H. Lashkari, “Arash habibi lashkari.” Disponible en https://www.cs.unb.ca/people/alashkar (Accedido el 20/01/2021).

[22] CIC and A. H. Lashkari, “Descarga de cicflowmeter.” Disponible en https://github.com/ahlashkari/CICFlowMeter (Accedido el 20/01/2021).

[23] J. A. Delgado, “Posible error detectado en cicflowmeter,” 2021. Disponible en https://github.com/ahlashkari/CICFlowMeter/issues/106 (Accedido el 25/01/2021).

[24] Kali, “Kali linux.” Disponible en https://www.kali.org/ (Accedido el 21/02/2021).

[25] R. Vinayakumar, “Repositiorio en github de r. vinayakumar,” 2018. Disponible en https://github.com/vinayakumarr/Network-Intrusion-Detection(Accedido el 15/09/2020).

[26] Sklearn, “Biblioteca sklearn.” Disponible en https://scikit-learn.org/stable/ (Accedido el 25/09/2020).

[27] Sklearn, “Función onehotencoder.” Disponible en https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (Accedido el 25/09/2020).

[28] Sklearn, “Función labelencoder.” Disponible en https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (Accedido el 27/09/2020).

[29] Sklearn, “Función train_test_split.” Disponible en https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (Accedido el 25/09/2020).

[30] J. L. Leevy and T. M. Khoshgoftaar, “A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data,” *Springer Open*, 2020. Disponible en https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00382-x.

[31] h00die, “adobe_pdf_embedded_exe payload,” 2019. Disponible en https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/exploit/windows/fileformat/adobe (Accedido el 23/02/2021).

[32] W. foundation, “Wireshark.” Disponible en https://www.wireshark.org/ (Accedido el 23/02/2021).

[33] scikitplot, “Matriz de confusión usando scikitplot.” Disponible en https://scikit-plot.readthedocs.io/en/stable/metrics.html#module-scikitplot.metrics (Accedido el 25/04/2021).

[34] multiprocessing, “Biblioteca multiprocessing.” Disponible en https://docs.python.org/3/library/multiprocessing.html (Accedido el 12/05/2021).

[35] W. McKinney, “Biblioteca pandas.” Disponible en https://pandas.pydata.org/ (Accedido el 25/09/2020).

[36] J. A. D. Soto, “Repositiorio en github del proyecto,” 2021. Disponible en https://github.com/javieradelgado/AnalisisDeTraficoCNN_LSTM(Accedido el 15/06/2021).

[37] R. Gomez, “Understanding categorical crossentropy loss,” 2018. Disponible en https://gombru.github.io/2018/05/23/cross_entropy_loss/ (Accedido el 15/09/2020).

[38] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[39] R. Gomez, “Los diferentes metodos de optimización en redes neuronales,” 2020. Disponible en https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5; (Accedido el 15/09/2020).

[40] J. Brownlee, “Gentle introduction to the adam optimization algorithm for deep learning,” 2017. Disponible en https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (Accedido el 15/09/2020).

[41] S. Saha, “A comprehensive guide to convolutional neural networks — the eli5 way,” 2018. Disponible en https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accedido el 12/08/2020).

[42] D. Calvo, “Red neuronal convolucional cnn,” 2017. Disponible en https://www.diegocalvo.es/red-neuronal-convolucional/ (Accedido el 18/08/2020).

[43] M. Venkatachalam, “Recurrent neural networks,” 2019. Disponible en https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce (Accedido el 13/08/2020).

[44] J. I. Garzón, “Red neuronal convolucional lstm,” 2018. Disponible en https://blog.gft.com/es/2018/11/06/como-usar-redes-neuronales-lstm-en-la-prediccion-de-averias-en-las-maquinas/ (Accedido el 18/08/2020).

[45] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, “Machine learning and deep learning methods for cybersecurity,” *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[46] G. Karatas, O. Demir, and O. Koray Sahingoz, “Deep learning in intrusion detection systems,” pp. 113–116, 2018.

[47] R. Abdulhammed, M. Faezipour, A. Abuzneid, and A. AbuMallouh, “Deep and machine learning approaches for anomaly-based intrusion detection of imbalanced network traffic,” *IEEE Sensors Letters*, vol. 3, no. 1, pp. 1–4, 2019.

Apéndices

A

Definiciones

A.1. Accuracy Paradox

Para comprender este concepto es necesario comprender primero la métrica accuracy, su explicación se puede encontrar en el apéndice F. Esta paradoja se explicará por medio de un ejemplo.

Se diseña un sistema de ML para detección de figuras geométricas en imágenes, que debe distinguir entre círculos y cuadrados. Se cuenta con una base de datos de 1000 muestras, 950 de ellas son círculos y 50 cuadrados.

Se selecciona un 70 % de los datos para train y 20 % para test. Esto hace que en las muestras de test contemos con 285 círculos y 15 cuadrados.

Después de entrenar el modelo, al testearlo se obtiene un accuracy de 95 %.

Cuando se pone el sistema en funcionamiento en una aplicación real, resulta que este no detecta imágenes con círculos. Esto puede parecer extraño porque el modelo tiene un accuracy del 95 %.

Usando la formula de accuracy, si comprobamos usando el total de los datos de test (300 muestras), obtenemos que un accuracy de 95 % representa 285 elementos bien clasificados. Si además extraemos la matriz de confusión del sistema, esta tendría el siguiente aspecto:

	Es circulo	Es cuadrado
Detectado circulo	0	0
Detectado cuadrado	15	285

Tabla A.1: Matriz de confusión de detección de formas geométricas

En la matriz de confusión se puede observar claramente que el sistema esta fallando, esta detectando todos los círculos como cuadrados. Pero como había una mayor de cantidad de cuadrados en los datos de test, y estos estaban bien clasificados, el sistema nos devolvía un gran accuracy.

Esta paradoja esta estrechamente ligada con el desbalanceamiento de los datos de test y de train, ya que si se produce este último, es más probable que se incurra en esta paradoja.

A.2. Categorical Crossentropy

Es un cálculo de perdida usado para clasificación multiclase. Es usada para definir una única clase como salida de entre todas las presentes en el modelo ya se obtiene en la capa de salida una probabilidad entre 0 y 1. En cada iteración se calcula la pérdida obtenida y se usa backpropagation para modificar los parámetros del modelo y así optimizarlos. Está muy relacionada con la función de activación softmax.

Las expresiones matemáticas se pueden consultar en [37].

A.3. Optimizador Adam

El optimizador Adam es un desarrollo del descenso de gradiente estocástico (SGD), diseñado para aplicarse a modelos de deep learning. Como todo optimizador, sirve para actualizar los valores de los diferentes parámetros de la red basándose en los datos de train.

Es un desarrollo de *Diederik Kingma y Jimmy Ba*, disponible en [38]. Según los autores, este optimizador tiene gran cantidad de beneficios entre los que se puede destacar:

- Eficiencia computacional.
- No requiere de mucha memoria.
- Recomendable para problemas con gran cantidad de parámetros y datos.
- Sencillez de implementación.

EN concreto, el optimizador Adam mezcla los beneficios tanto de *AdaGrad* (Adaptive Gradient Algorithm) como *RMSProp* (Root Mean Square Propagation). Estos dos optimizadores tambien son modificaciones de *SGD*.

- **AdaGrad** modifica el factor de entrenamiento, en SGD se consideraba un único valor que englobaba a todos los parámetros entrenables del sistema. AdaGrad asigna un factor de entrenamiento diferente para cada parámetro pero, como es inviable calcular continuamente un valor determinado para cada parámetro, este factor se obtiene escalando y adaptando el gradiente de la iteración anterior respecto de cada dimensión.
- **RMSProp** también mantiene el entrenamiento por parámetro, pero en este caso, estos factores se adaptan basandose en en la media de declive exponencial de los gradientes de descenso anteriores al cuadrado.

Al mezclar el funcionamiento de estos dos optimizadores, Adam genera dos valores denominados beta1 y beta2 para controlar el aprendizaje de cada parámetro. Estos valores idealmente deben ser próximos a 1 en un principio, generalmente beta1=0.9 y beta2=0.999.

También incluye otros dos parámetros, α, que representa el ritmo de aprendizaje, idealmente $\alpha = 0{,}001$. El segundo parámetro es ε, su finalidad es prevenir divisiones por cero en los cálculos, por eso se le asigna un valor muy pequeño, $\varepsilon = 10^{-8}$.

Con estas modificaciones, Adam es capaz de minimizar las pérdidas para modelos de clasificación multiclase como se puede observar en la siguiente imagen:

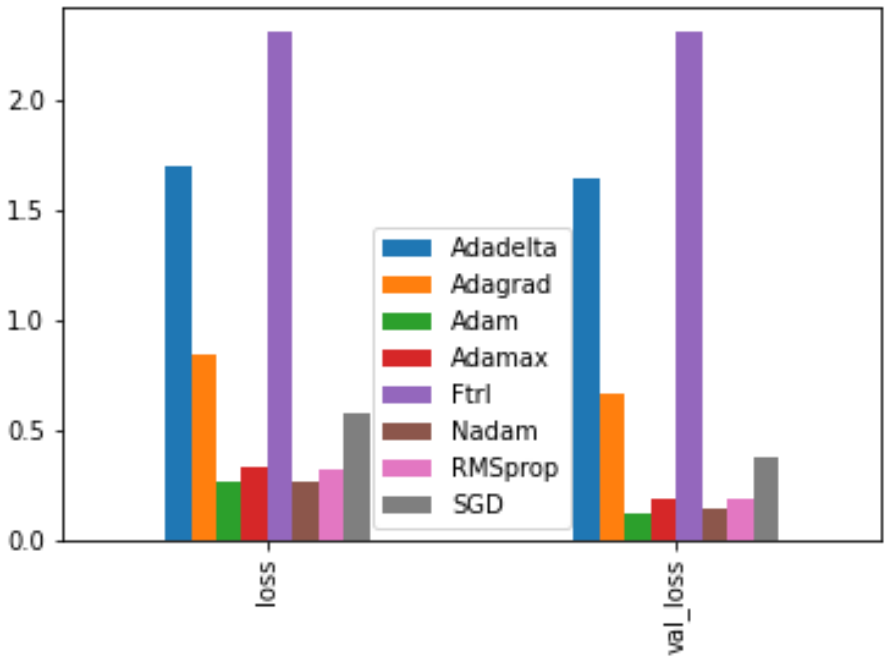

Figura A.1: Comparativa de optimizadores en clasificación multiclase [39]

Información extraida de [40].

B

Machine Learning

Tiene como meta generar u optimizar sistemas de tratamiento de datos. Para lograr esta meta, los sistemas de Machine Learning están estrechamente relacionados con campos como la estadística y la informática. Se basa en la estadística para la toma de decisiones a partir de reconocimiento de patrones de los datos de entrada y se basa en la informática para generar los diferentes algoritmos que nos permiten resolver los diferentes problemas, no solo se basa en la informática para esto, si no que también está estrechamente ligado con otros campos relacionados con este último como lo pueden ser las diferentes tecnologías aplicadas a GPUs para aceleración de modelos o los diferentes ISAs disponibles en las CPUs.

Esta tecnología está en pleno crecimiento a día de hoy. Es posible que el usuario medio no se haya dado cuenta de las diferentes aplicaciones, pero muchos sistemas, como las recomendaciones de Netflix o algunos IDS (Intrusion Detection System), están basados en esta tecnología. El funcionamiento de los algoritmos de Machine Learning depende de la aproximación que se esté usando. Hay gran variedad de aproximaciones, desde intentar “replicar” el funcionamiento de un cerebro (Redes Neuronales) a ir “podando” un árbol de decisión para ir descartando variables no importantes.

Hay 3 tipos básicos de entrenamiento de algoritmos de Machine Learning; entrenamiento supervisado, entrenamiento no supervisado y entrenamiento híbrido. Generalmente el entrenamiento de un algoritmo de ML consiste en ejecutarlo sobre un conjunto de datos que puede tener diferentes formas, dependiendo de lo que se pretenda que haga este algoritmo, estos datos pueden ser imágenes, texto, audio, etc.

- **Entrenamiento supervisado**: Los datos de entrada del modelo (también llamado dataset), están etiquetados. Que estén etiquetados implica que, para cada entrada, se ha definido en el dataset la salida que debería tener el sistema. El principal problema de este entrenamiento es que no es posible aplicarlo cuando no se está seguro de los datos de entrada o de las relaciones entre estos, también hay que tener en cuenta posibles errores humanos a la hora de etiquetar cada flujo de entrada.
- **Entrenamiento supervisado**: En este tipo de entrenamiento los datos no están etiquetados, por lo tanto, el dataset no incluye las salidas para los datos de entrada. Esto tiene muchos beneficios, pero también alguna desventaja. El principal beneficio de este tipo de entrenamiento son que el modelo tiene la posibilidad de descubrir patrones ocultos o agrupaciones de datos que el usuario no ha considerado importantes, pero en realidad si lo eran. También es capaz de eliminar los problemas de origen humano.

- **Entrenamiento semi-supervisado o híbrido**: Este tipo de entrenamiento se encuentra entre los dos definidos anteriormente. Los datos de entrada se pueden dividir en dos subgrupos, datos etiquetados con su salida correspondiente y datos sin etiquetar. La relación de estos dos tipos suele ser la siguiente: un pequeño porcentaje del dataset se corresponde con datos etiquetados y un gran porcentaje se corresponde con datos sin etiquetar. Este tipo de entrenamiento nos permite obtener beneficios de los dos tipos anteriores y limitar las desventajas.

C

Redes Neuronales

C.1. Elementos básicos

Una red neuronal consta de 3 elementos básicos, a partir de estos se genera una estructura compleja.

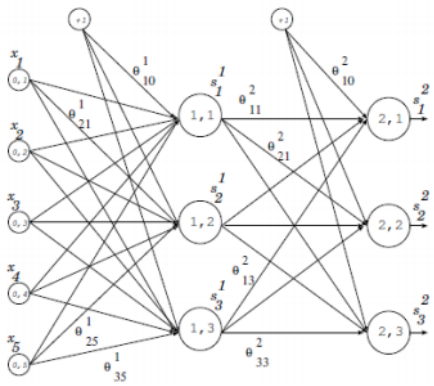

Figura C.1: Elementos básicos de una red neuronal

- **Neuronas**: Son las unidades de procesamiento y pueden tener diferentes funciones de activación. Una función de activación define la operación que lleva a cabo cada neurona para generar una salida.
- **Pesos**: Estos son valores que se les asigna a cada conexión entre 2 neuronas y que luego se computan junto con el valor de salida de la neurona inicial en la neurona que tiene esa conexión como entrada. Este parámetro es ajustable durante el entrenamiento.
- **Umbral**: Es un valor que se suele restar y que sirve para tener cierto control sobre la operación realizada. También es ajustable durante el entrenamiento.

Esto hace que la operación llevada a cabo en cada neurona sea la siguiente:

$$output = FA(\sum_{i=1}^{d} w_j * x_j + b) \tag{C.1}$$

donde FA representa la función de activación de la neurona, w el peso de una conexión, x el output

de una neurona y b el bias.

Redes Neuronales Convolucionales

Las CNN son NN especializadas en tratamiento de imágenes [2], ya que su funcionamiento se basa en la realización de convoluciones a los datos de entrada [2]. Se pueden aplicar tanto a matrices unidimensionales como bidimensionales. Esta última característica es la que convierte a esta red en idónea para tratamiento de imágenes [41].

Estas capas convolucionales se mezclan con otros tipos de capa de manera que, a medida que se va avanzando en esta red, las dimensiones de las capas disminuyen [2] [41].

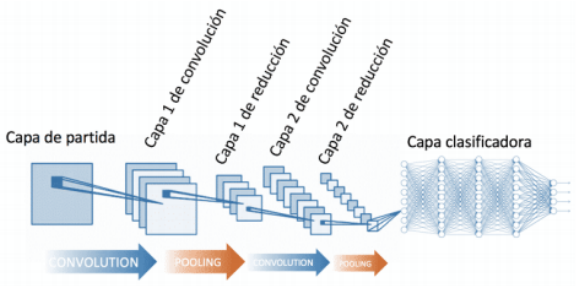

Figura C.2: Estructura de ejemplo de CNN [42]

Al reducir las dimensiones en las últimas capas y aplicar convoluciones se consigue disminuir el coste computacional de la red, pero, como contra-parte, las neuronas de las últimas pierden sensibilidad frente a modificaciones de los datos de entrada [41].

Redes Neuronales Recurrentes

Esta clase de redes neuronales no tiene una estructura de capas estándar [43]. Se crean ciclos de retroalimentación en cada neurona o entre neuronas que permiten conseguir temporalidad y evaluar datos de entrada en instantes de tiempo posteriores [41].

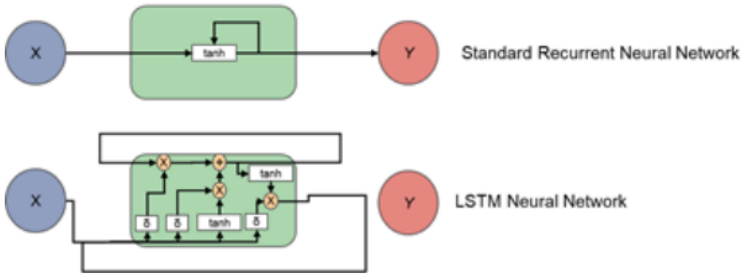

Figura C.3: Estructura de LSTM [44]

En la imagen superior se pueden observar dos métodos diferentes para proveer de una estructura

temporal a una neurona. El método que se usará en este trabajo es el segundo, LSTM [44].

Para aplicaciones de detección de ataques este tipo de redes a primera vista parece muy recomendable debido a su capacidad de análisis temporal.

D

IDSs

Existen tres tipos de IDS y los 3 son buenos candidatos para modelos de ML. Tienen cierta similitud con las formas de entrenamiento. Están los IDS basados en firma, que sirven para detectar ataques previamente conocidos; los IDS basados en anomalías, que sirven para detectar anomalías de tráfico y así poder detectar ataques de day-zero (ataques nuevos de los que no se tiene constancia) ; y por último, IDS híbridos, que mezclan ambos conceptos.

Estos sistemas pueden basar su entrenamiento en datasets propios de las empresas que los crean o en datasets públicos. El mayor problema al que se enfrenta un particular si desea entrenar un modelo de Machine Learning para uso propio es que los datasets son muy difíciles de generar, por lo tanto, solo hay limitados de acceso público, que por lo general, suelen estar ligeramente desfasados. Esto se debe a que el mundo del cibercrimen y la ciberseguridad se mueve muy rápido y se crean muchos ataques nuevos al año, por lo tanto, tener siempre un dataset actualizado es un trabajo muy arduo, solo posible para corporaciones.

Algunos de los planteamientos más comunes para estos modelos de detección de ciberataques son SVM, arboles de decisión, y arquitecturas híbridas o no de redes neuronales convolucionales y recurrentes. Son de especial interés las redes neuronales recurrentes ya que, como se ha comentado, permiten cierta dependencia temporal. Tanto en [45] como en [46] se hace una breve explicación de los diferentes modelos aplicables y datasets públicos más comunes. En [47] se explican diferentes aproximaciones de machine learning para un IDS.

E

Modelo propuesto por Vinayakumar R.

En esta investigación se propone una arquitectura consistente de una red neuronal convolucional unida a una red neuronal recurrente (LSTM):

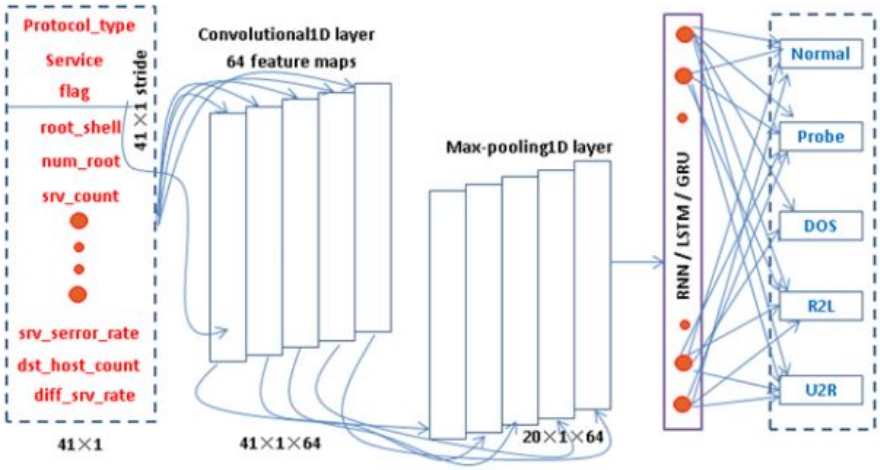

Figura E.1: Modelo propuesto

En esta investigación, aparte de proponer este modelo, Vinayakumar explora otras posibilidades para la arquitectura y en todas obtiene unos resultados extremadamente buenos. Se puede observar que hay 2 modelos que obtienen resultados ligeramente mejores sin necesidad de utilizar una red LSTM. El alumno ha decidido basarse en el modelo que contiene una LSTM debido al hecho de que gran parte de los ciberataques actuales tienen dependencia temporal de paquetes entre si, y esta red permitirá un mayor control sobre dependencias entre paquetes.

Vinayakumar usa el conjunto de datos KDD CUP 99', lo que hace que debamos adaptar los tamaños de los elementos de la red a los que se muestran en la imagen anterior, estos parámetros se verán ligeramente modificados cuando se use un nuevo dataset.

Le estructura planteada por Vinayakumar es la siguiente (la capa superior representa la entrada, en adelante será referida como capa 1):

Este modelo cuenta con 9 capas organizadas de la siguiente manera (Se representa LSTM como

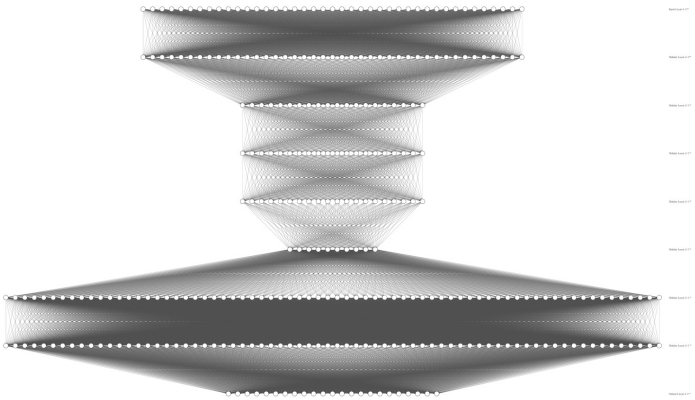

Figura E.2: Capas de modelo propuesto

una sola capa por simplicidad):

- **Capas 1 y 2**: Son capas convolucionales 1D que nos permiten hacer un primer filtrado de los datos.
- **Capa 3**: Es una capa de MaxPooling 1D usada para seleccionar el máximo de cada 2 elementos de la capa anterior.
- **Capas 4 y 5**: Son capas convolucionales 1D que nos permiten volver a filtrar.
- **Capa 6**: Otra capa de MaxPooling 1D usada para volver a seleccionar el máximo de cada 2 elementos de la capa anterior.
- **Capa 7**: Esta capa representa la red LSTM, solo se ha representado la salida de esta unida a la salida de las capas anterior y posterior.
- **Capa 8**: Una capa de Dropout usada para prevenir overfitting del modelo.
- **Capa 9**: Una capa Dense para obtener los valores de salida del modelo.

En total esta red cuenta con 143.945 parámetros, de los cuales todos son entrenables. Todos los nodos de la red usan como función de activación la función **relu**, salvo las capas de MaxPooling 1D cuyos nodos no tienen función de activación, la red LSTM cuyas neuronas que usa una función de activación **tanh** y la capa de salida que usa **softmax** porque esta función mapea la salida de forma que la suma de los valores de todos los nodos es 1, lo que la convierte en un distribuidor de probabilidades de tráfico perfecto para un problema de clasificación multiclase.

Cada uno de estos parámetros representa el peso asociado a una conexión de entrada a una neurona de la capa o el bias asociado a una neurona en concreto. Es lógico que estos parámetros solo se encuentren en capas donde se ejecute alguna operación como una convolución o activación de las neuronas como es la última capa y no en capas intermedias de maxpooling o dropout ya que por la propia labor de estas capas no es necesario que cuenten con parámetros. Un caso especial a comentar es el de la red LSTM identificada como la capa 7, se observa que este elemento cuenta con la mayor cantidad de parámetros debido a que representa una red entera.

Capa	Parámetros
1	256
2	12.352
3	0
4	24.704
5	49.280
6	0
7	55720
8	0
9	1.633
Total	143.945

Tabla E.1: Número de parámetros por capa

E.1. Total de paquetes por tipo

```
smurf.              2807886
neptune.            1072017
normal.              972781
satan.                15892
ipsweep.              12481
portsweep.            10413
nmap.                  2316
back.                  2203
warezclient.           1020
teardrop.               979
pod.                    264
guess_passwd.            53
buffer_overflow.         30
land.                    21
warezmaster.             20
imap.                    12
rootkit.                 10
loadmodule.               9
ftp_write.                8
multihop.                 7
phf.                      4
perl.                     3
spy.                      2
```

Figura E.3: Número de paquetes por tipo en KDDCUP 99'

F

Estudio de Precision, Recall y F1 Score

Precisión es un parámetro que sirve para medir la calidad de un modelo, en si su finalidad es mostrar el porcentaje de tráficos de una clase correctamente predichos frente a los tráficos de otras clases predichos como esa. Es decir, de todos los paquetes/flujos predichos como la clase X, cuantos son de verdad de la clase X. Es una medición correcta de los positivos y sigue la siguiente formula:

$$Precisión = \frac{TP}{TP + FP} \tag{F.1}$$

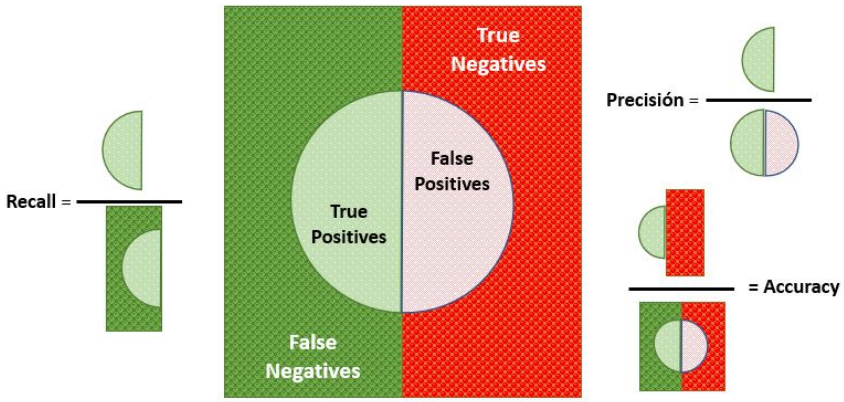

Figura F.1: Calculo de parámetros para la evaluación del modelo

La imagen superior es una representación gráfica de la extracción de los diferentes parámetros donde TP se corresponde con True Positives, False Positives es FP, FN es False Negatives y True Negatives se corresponde con TN. Todos estos parámetros están referidos a clasificación biclase pero están igualmente definidos para clasificación multiclase. En la imagen también se puede observar el calculo de los principales parámetros, esto se ha añadido para que unido a las diferentes explicaciones de cada parámetro, estos sean más fáciles de comprender.

Recall es un parámetro que nos devuelve información acerca de la cantidad de elementos que el modelo es capaz de clasificar correctamente de todos los que se deberían clasificar como esa clase. Si este parámetro es muy alto, nos indicará que se han clasificado correctamente la mayor parte de los

elementos de la clase.

$$Recall = \frac{TP}{TP + FN} \tag{F.2}$$

El parámetro **Accuracy/Exactitud** sirve para medir la cantidad de veces que acierta el modelo, es el parámetro que más se suele usar pero es altamente engañoso. Para cuantificar la proporción de aciertos del modelo se tiene en cuenta tanto los elementos correctamente clasificados negativos como positivos.

$$Accuracy/Exactitud = \frac{TP + TN}{TP + TN + FP + FN} \tag{F.3}$$

F1-Score es un parámetro que sirve para evaluar de manera conjunta los parámetros de precisión y recall, esto simplifica la evaluación del rendimiento del modelo. Este parámetro F1 es solo una opción de todas las posibles para el parámetro F_B donde B sirve para variar la ponderación total de cada uno de los parámetros.

$$F_B = (1 + B^2)\frac{precision * recall}{(B^2 * precision) + recall} \tag{F.4}$$

Eso hace que el parámetro aplicado en este modelo sea el siguiente valor:

$$F_1 = 2\frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FN + FP)} \tag{F.5}$$

Todos estos parámetros se calculan para todas las clases para evaluar el funcionamiento del modelo para clase en particular. Además, se calculan también para el modelo entero, para este fin nos valemos del macro average y el micro average.

El cálculo de **macro average** es un computo independiente de las características de cada clase, no tiene en cuenta la diferencia de tamaño entre estas y las evalúa a todas por igual, lo que hace que ofrezca una visión un poco sesgada de la funcionalidad del modelo.

De manera contraria, **micro average** o **weighted average** si que tiene en cuenta la contribución de cada clase al conjunto de datos de test. El parametro del calculo de la media asociado a cada clase esta multiplicado por el porcentaje que representa esa clase dentro del conjunto.

G

Resultados por clase en replicación

En este anexo se encontrarán los datos que no se han podido mostrar a la vez que los datos globales al comentar los resultados obtenidos de la replicación de la investigación debido a la falta de espacio. A continuación se mostrarán los valores de los parámetros comentados en el capitulo de pruebas para cada una de las clases:

	Precisión	Recall	F1-Score	support
smurf	0.99995020	0.99998862	0.99996941	70.2776
neptune	0.99987676	0.99995518	0.99991597	267.748
normal	0.99882669	0.99912284	0.99897474	242.830
satan	0.99871564	0.98281092	0.99069945	3.956
ipsweep	0.99433522	0.97072219	0.98238683	3.074
portsweep	0.98805257	0.98530580	0.98667727	2.518
nmap	0.79881657	0.95575221	0.87026591	565
back	0.94953271	0.98449612	0.96669838	516
warezclient	0.91780822	0.78823529	0.84810127	255
teardrop	0.98804781	0.99200000	0.99001996	250
pod	0.93548387	0.42028986	0.58000000	255
guess_passwd	0.81250000	0.92857143	0.86666667	14
buffer_overflow	0.0	0.0	0.0	13
land	1	0.16666667	0.28571429	6
ftp_write	0.0	0.0	0.0	4
loadmodule	0.0	0.0	0.0	4
warezmaster	0.0	0.0	0.0	3
multihop	0.0	0.0	0.0	2
rootkit	0.0	0.0	0.0	2
spy	0.0	0.0	0.0	2
imap	0.0	0.0	0.0	1

Tabla G.1: Parámetros específicos de replicación

Y la matriz de confusión normalizada será la siguiente:

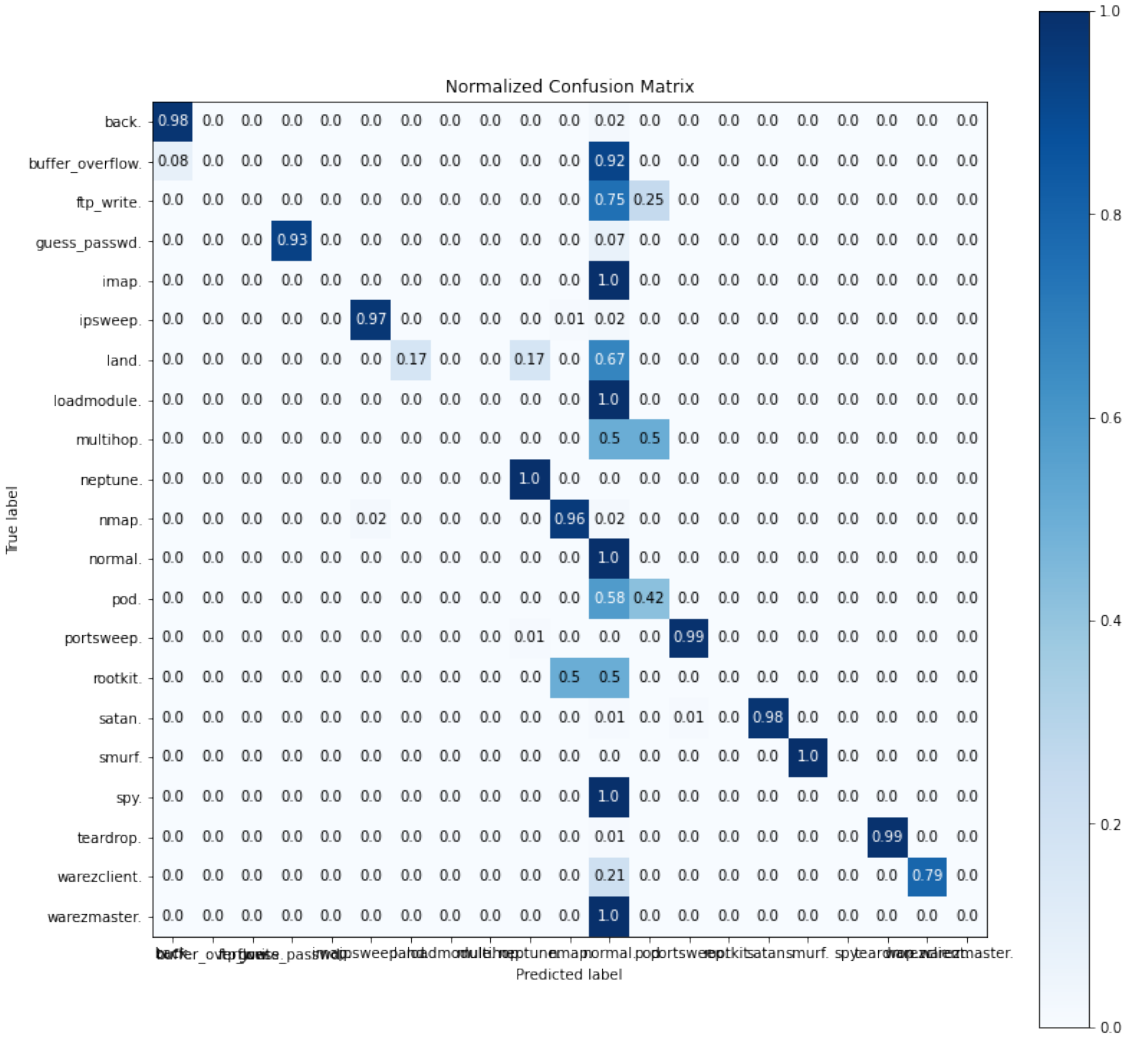

Figura G.1: Matriz de confusión normalizada para modelo sobre KDD

H

Resultados por clase en desarrollo sobre CSE-CIC-IDS2018 con Timestamp

En este anexo se encontrarán los datos que no se han podido mostrar a la vez que los datos globales al comentar los resultados obtenidos de el estudio por clases de los diferentes tipos de ataque encontrados en CSE-CIC-IDS2018. A continuación se mostrarán los valores de los parámetros comentados en el capitulo de pruebas para cada una de las clases:

H.1. Ataques de Fuerza Bruta

	Precisión	Recall	F1-Score	support
Benigno	0.99979721	0.99974128	0.99976924	429.033
FTP-BruteForce	0.99803751	0.99993791	0.99898681	48.316
SSH-BruteForce	0.99972237	0.99935955	0.99954093	242.830
Brute Force -Web	0.75675676	0.40579710	0.52830189	69
Brute Force -XSS	1.0	0.5	0.66666667	18
SQL Injection	0.0	0.0	0.0	10

Tabla H.1: Parámetros específicos de Fuerza Bruta sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

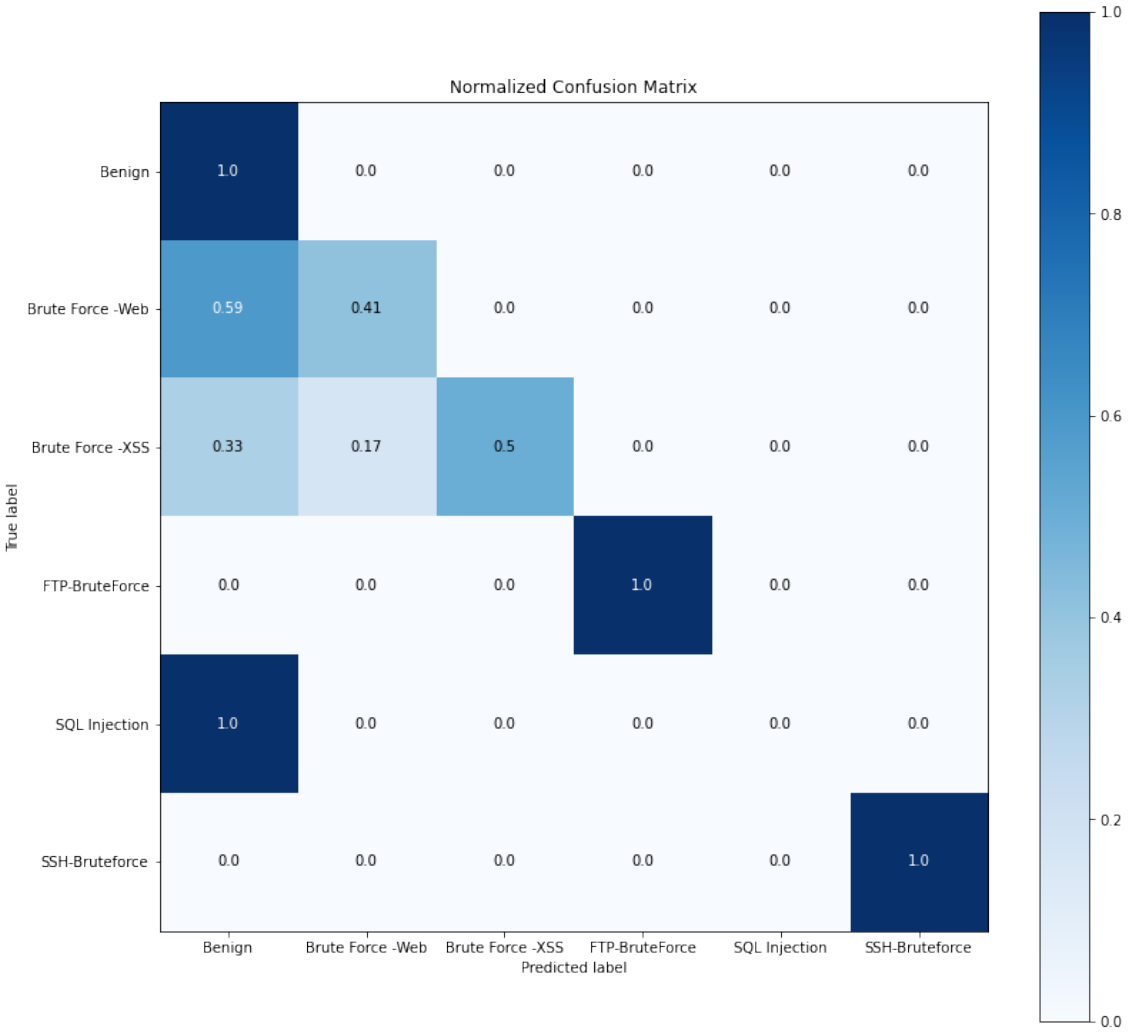

Figura H.1: Matriz de confusión normalizada para ataques de Fuerza Bruta sobre CSE-CIC-IDS2018

H.2. Ataques de Infiltración

	Precisión	Recall	F1-Score	support
Benigno	0.86818283	0.98253087	0.92182430	195.316
Infiltración	0.77256366	0.28457780	0.41594143	40.727

Tabla H.2: Parámetros específicos de Infiltración sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

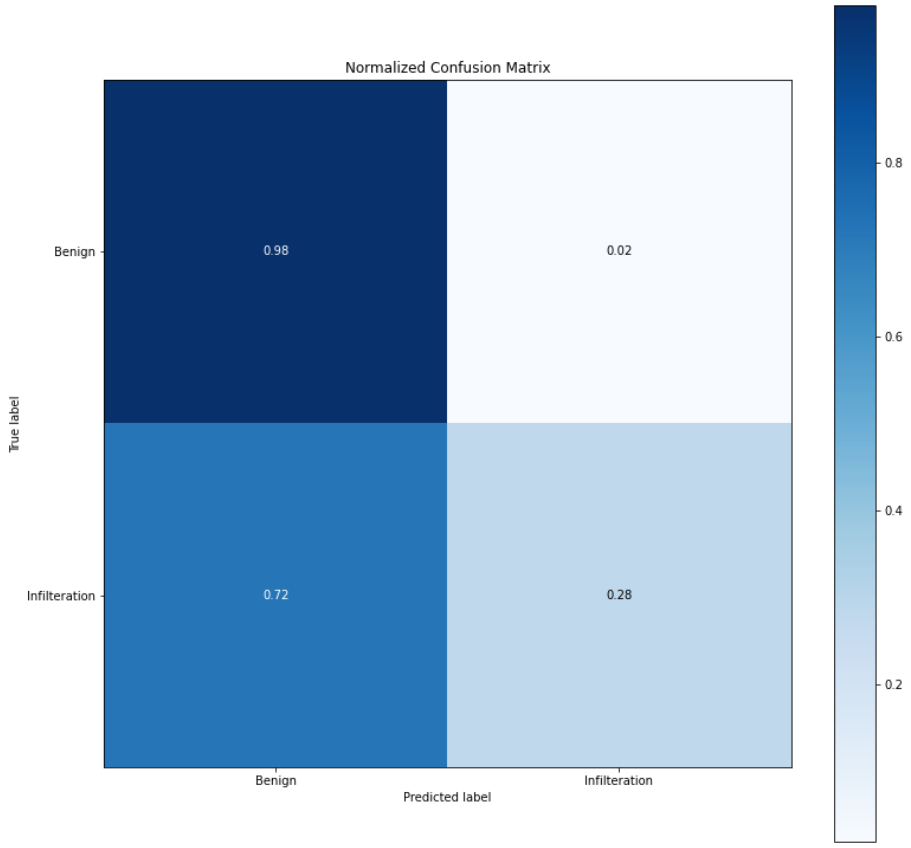

Figura H.2: Matriz de confusión normalizada para ataques de Infiltración sobre CSE-CIC-IDS2018

H.3. Ataques de DoS/DDoS

	Precisión	Recall	F1-Score	support
Benigno	0.99862149	0.99889157	0.99875651	451.090
DDOS attack-HOIC	0.99995335	0.99977846	0.99986590	171.530
DoS attacks-Hulk	0.99976577	0.99671343	0.99823726	115.622
DoS attacks-SlowHTTPTest	0.99853073	0.99910640	0.99881848	34.691
DoS attacks-GoldenEye	0.97339864	0.98647202	0.97989172	10.275
DoS attacks-Slowloris	0.94313454	0.97947425	0.96096096	2.777
DDOS attack-LOIC-UDP	1.0	0.99103139	0.99549550	446

Tabla H.3: Parámetros específicos de DoS/DDoS sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

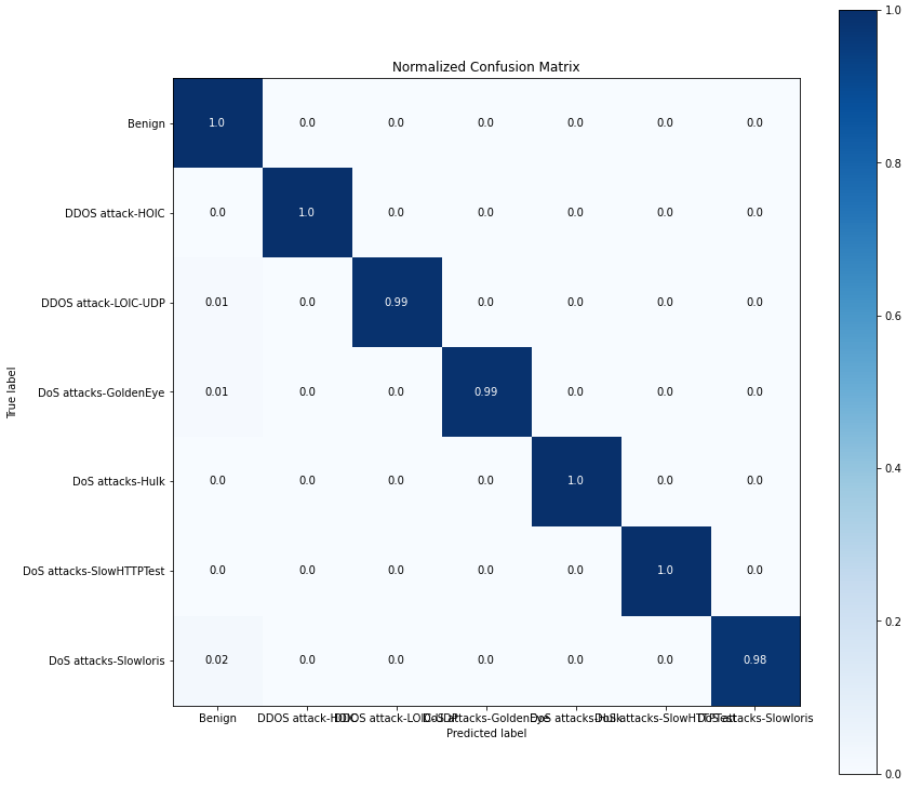

Figura H.3: Matriz de confusión normalizada para ataques de DoS/DDoS sobre CSE-CIC-IDS2018

Resultados por clase en desarrollo sobre CSE-CIC-IDS2018 sin Timestamp

En este anexo se encontrarán los datos que no se han podido mostrar a la vez que los datos globales al comentar los resultados obtenidos de el estudio por clases de los diferentes tipos de ataque encontrados en CSE-CIC-IDS2018 eliminando la columna del parametro TimeStamp. A continuación se mostrarán los valores de los parámetros comentados en el capitulo de pruebas para cada una de las clases:

I.1. Ataques de Fuerza Bruta

	Precisión	Recall	F1-Score	support
Benigno	0.99973659	0.99963173	0.99968416	429.033
FTP-BruteForce	0.99847073	0.99997930	0.99922445	48.316
SSH-BruteForce	0.99810243	0.99938090	0.99874125	242.830
Brute Force -Web	1.0	0.01449275	0.02857143	69
Brute Force -XSS	1.0	0.44444444	0.61538462	18
SQL Injection	0.0	0.0	0.0	10

Tabla I.1: Parámetros específicos de Fuerza Bruta sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

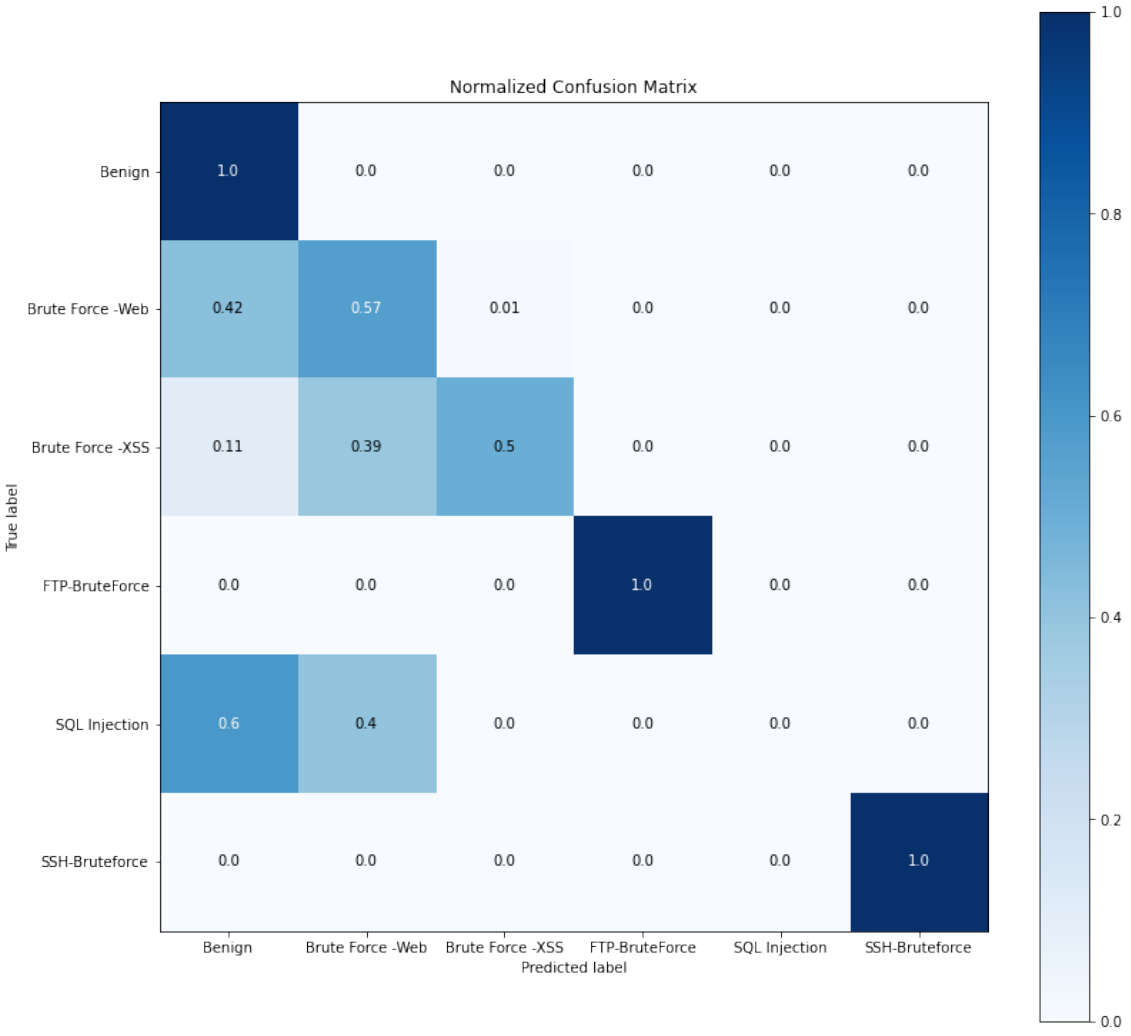

Figura I.1: Matriz de confusión normalizada para ataques de Fuerza Bruta sobre CSE-CIC-IDS2018

I.2. Ataques de Infiltración

	Precisión	Recall	F1-Score	support
Benigno	0.82745940	1.0	0.90558444	195.316
Infiltración	0.0	0.0	0.0	40.727

Tabla I.2: Parámetros específicos de Infiltración sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

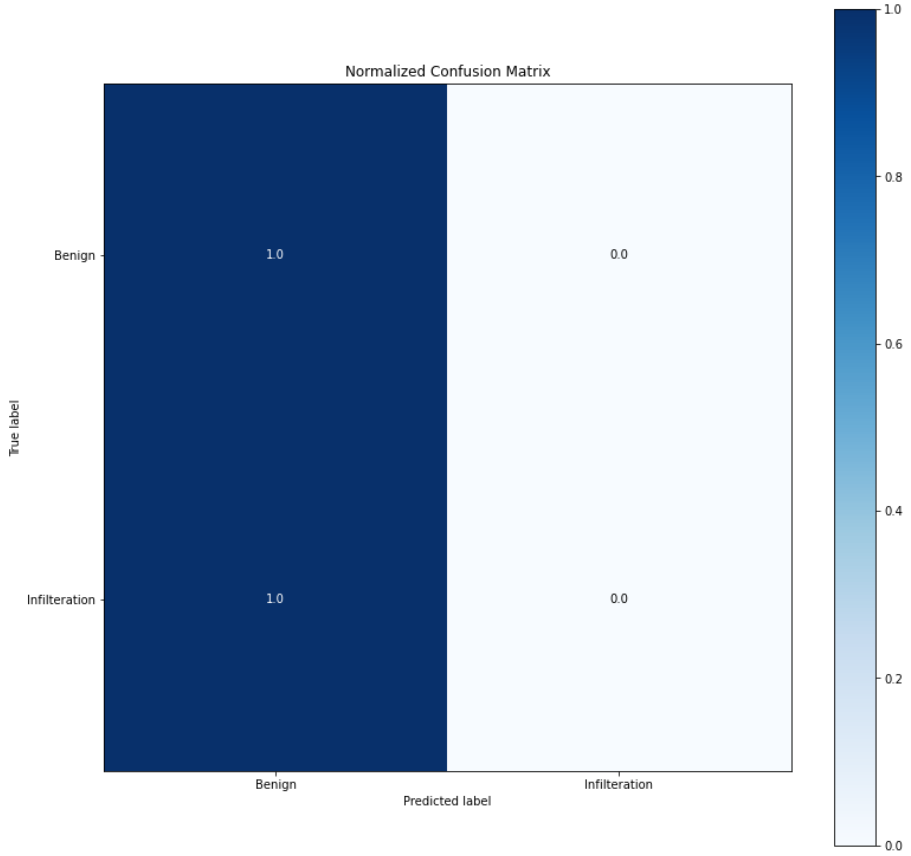

Figura I.2: Matriz de confusión normalizada para ataques de Infiltración sobre CSE-CIC-IDS2018

I.3. Ataques de DoS/DDoS

	Precisión	Recall	F1-Score	support
Benigno	0.99916600	0.99860560	0.99888572	451.090
DDOS attack-HOIC	0.99948141	0.99999417	0.99973772	171.530
DoS attacks-Hulk	0.99884915	0.99837401	0.99861153	115.622
DoS attacks-SlowHTTPTest	0.99879025	0.99956761	0.99917878	34.691
DoS attacks-GoldenEye	0.97077922	0.98939173	0.97999711	10.275
DoS attacks-Slowloris	0.94706518	0.94706518	0.94706518	2.777
DDOS attack-LOIC-UDP	0.99773756	0.98878924	0.99324324	446

Tabla I.3: Parámetros específicos de DoS/DDoS sobre CSE-CIC-IDS2018

Y la matriz de confusión normalizada será la siguiente:

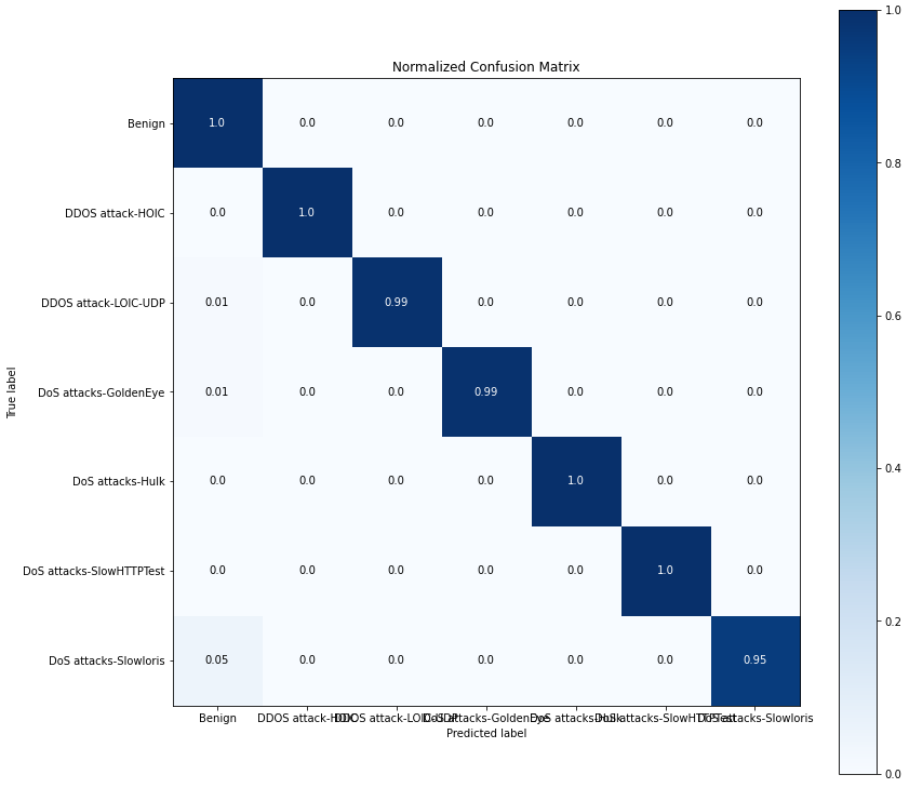

Figura I.3: Matriz de confusión normalizada para ataques de DoS/DDoS sobre CSE-CIC-IDS2018

J

Estudio IDS con arquitectura global

En este anexo se encontrarán los datos que no se han podido mostrar a la vez que los datos globales al comentar los resultados obtenidos del estudio de la arquitectura global pa:

	Precisión	Recall	F1-Score	support
Benigno	0.99983218	0.99982519	0.99982868	429.033
FTP-BruteForce	0.99803751	0.99993791	0.99898681	48.316
SSH-BruteForce	0.99972237	0.99935955	0.99954093	242.830
Brute Force -Web	0.75675676	0.40579710	0.52830189	69
Brute Force -XSS	1.0	0.5	0.66666667	18
SQL Injection	0.0	0.0	0.0	10
DDOS attack-HOIC	0.99995335	0.99977846	0.99986590	171.530
DoS attacks-Hulk	0.99976577	0.99671343	0.99823726	115.622
DoS attacks-SlowHTTPTest	0.99853073	0.99910640	0.99881848	34.691
DoS attacks-GoldenEye	0.97339864	0.98647202	0.97989172	10.275
DoS attacks-Slowloris	0.94313454	0.97947425	0.96096096	2.777
DDOS attack-LOIC-UDP	1.0	0.99103139	0.99549550	446
Infiltración	0.51700680	0.01866084	0.03602152	40.727

Tabla J.1: Parámetros específicos por clase en arquitectura global

Y la matriz de confusión normalizada será la siguiente:

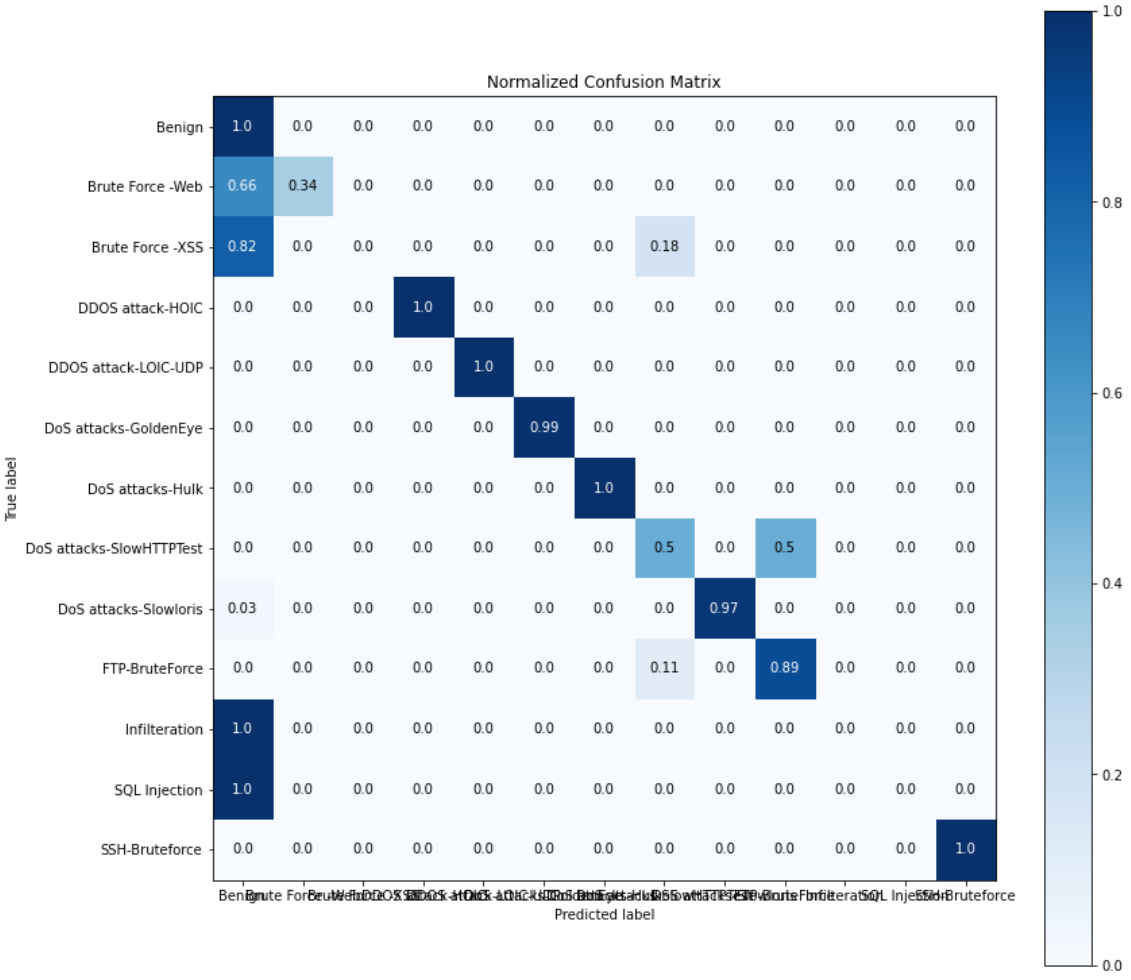

Figura J.1: Matriz de confusión normalizada para ataques detectados por arquitectura global

K

Estudio IDS con arquitectura en paralelo sin LSTM

En este anexo se encontrarán los datos que no se han podido mostrar a la vez que los datos globales al comentar los resultados obtenidos de el estudio de la aplicación del modelo sin red LSTM. A continuación se mostrarán los valores de los parámetros comentados en el capitulo de pruebas para cada una de las clases:

K.1. Ataques de Fuerza Bruta

	Precisión	Recall	F1-Score	support
Benigno	0.99979488	0.99977857	0.99978673	429.033
FTP-BruteForce	0.99845003	0.99993791	0.99919341	48.316
SSH-BruteForce	0.99946647	0.99980786	0.99963714	242.830
Brute Force -Web	0.0	0.0	0.0	69
Brute Force -XSS	1.0	0.83333333	0.90909091	18
SQL Injection	0.0	0.0	0.0	10

Tabla K.1: Parámetros específicos de Fuerza Bruta para arquitectura en paralelo sin LSTM

Y la matriz de confusión normalizada será la siguiente:

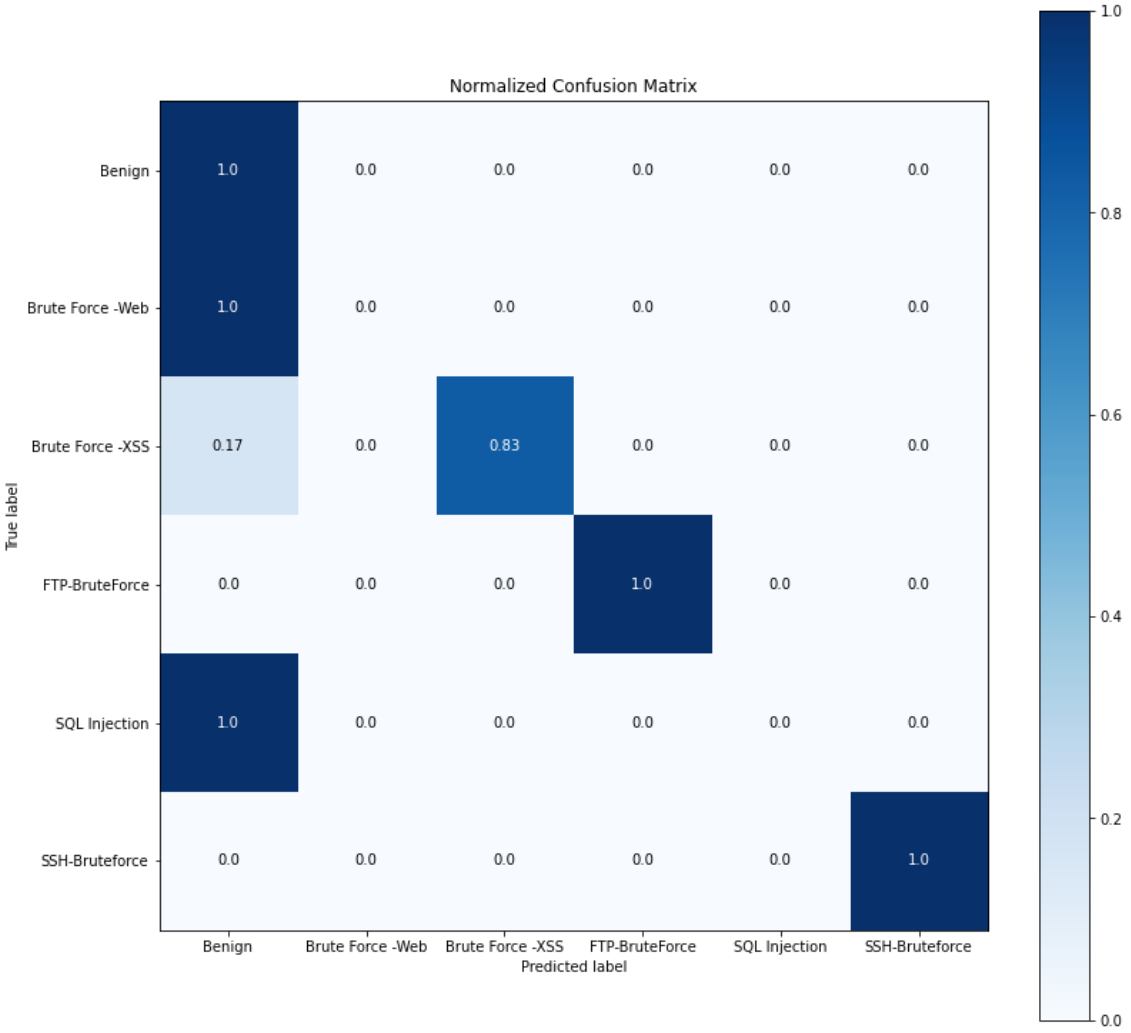

Figura K.1: Matriz de confusión normalizada para ataques de Fuerza Bruta para arquitectura en paralelo sin LSTM

K.2. Ataques de Infiltración

	Precisión	Recall	F1-Score	support
Benigno	0.82928918	0.99651334	0.90524328	195.316
Infiltración	0.49254844	0.01623002	0.03142456	40.727

Tabla K.2: Parámetros específicos de Infiltración para arquitectura en paralelo sin LSTM

Y la matriz de confusión normalizada será la siguiente:

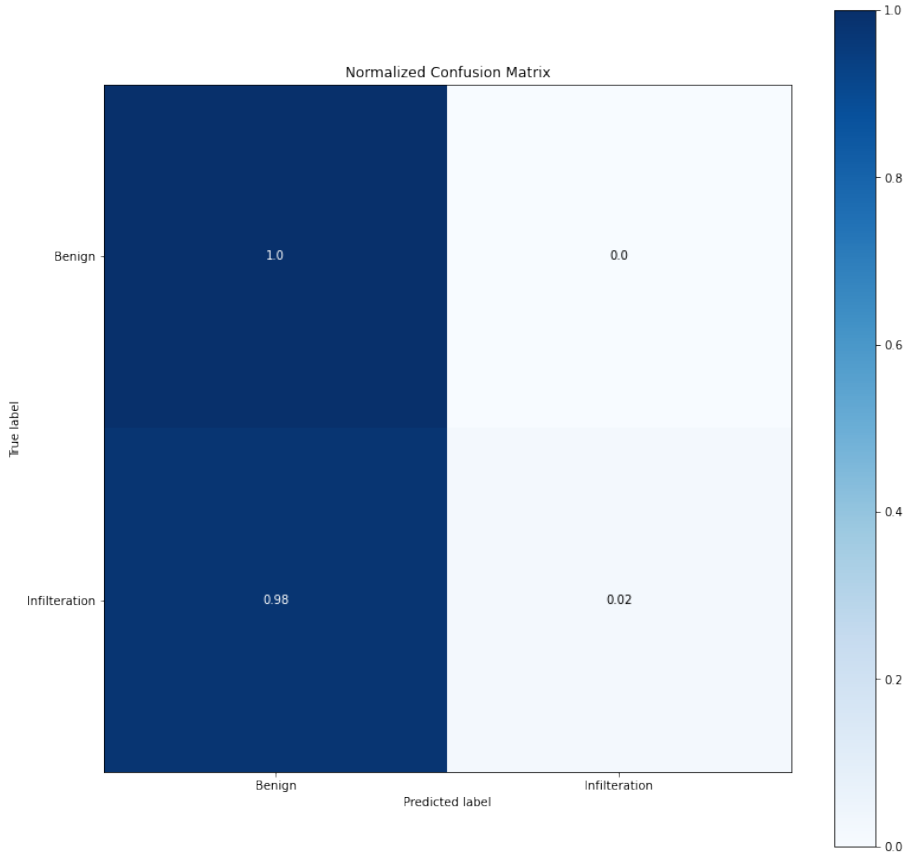

Figura K.2: Matriz de confusión normalizada para ataques de Infiltración para arquitectura en paralelo sin LSTM

K.3. Ataques de DoS/DDoS

	Precisión	Recall	F1-Score	support
Benigno	0.99966269	0.99862777	0.99914496	451.090
DDOS attack-HOIC	0.99983674	0.99972017	0.99977845	171.530
DoS attacks-Hulk	0.99992213	0.99959350	0.99975779	115.622
DoS attacks-SlowHTTPTest	0.99893458	1.0	0.99946700	34.691
DoS attacks-GoldenEye	0.96035985	0.99737226	0.97851618	10.275
DoS attacks-Slowloris	0.95053988	0.98271516	0.96635977	2.777
DDOS attack-LOIC-UDP	1.0	0.99551570	0.99775281	446

Tabla K.3: Parámetros específicos de DoS/DDoS para arquitectura en paralelo sin LSTM

Y la matriz de confusión normalizada será la siguiente:

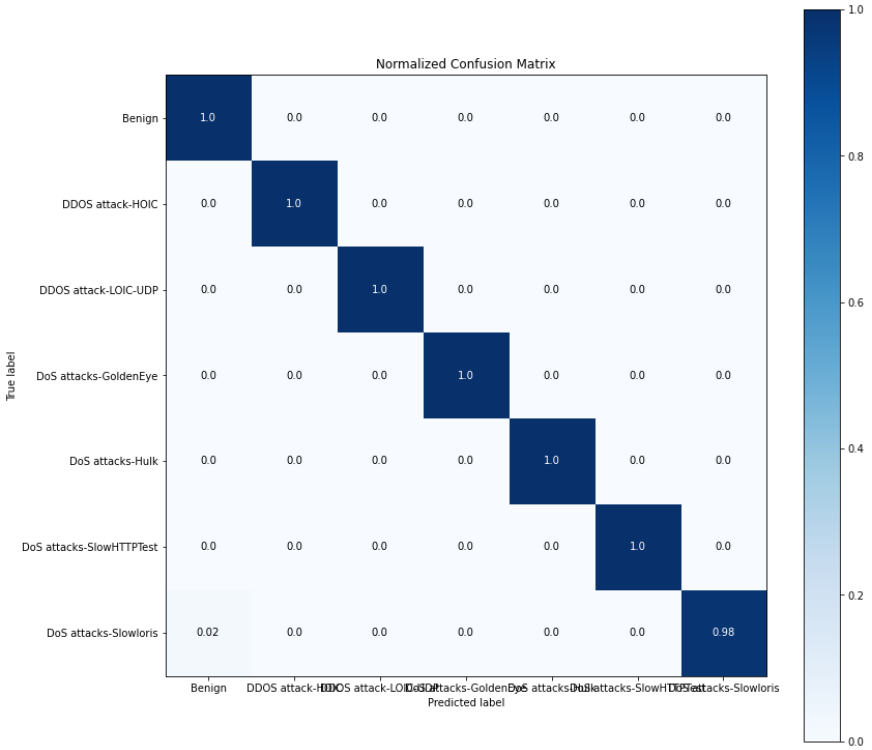

Figura K.3: Matriz de confusión normalizada para ataques de DoS/DDoS para arquitectura en paralelo sin LSTM

L

Códigos

L.1. Generación de sistema paralelo CSE-CIC-IDS2018

Código L.1: Código usado para generar ejecución en paralelo

```
processes = []
    for i in range(3):
        p = multiprocessing.Process(target=Prediccion,args=(i,input_NN,networks[i]))
        p.start()
        processes.append(p)

    # Now you can wait for the networks to finish training before executing the
    # rest of the script
    for process in processes:
        process.join()
```

L.2. Definición de la función Prediccion()

Código L.2: Función Prediccion() usada para ejecutar todas las redes

```
def Prediccion(option, input_N, ruta):
    if (option==0):
        #Cargamos los resultados obtenidos en la etapa de train
        print("loading Infilteration weigths...")
        Infilteration.load_weights(ruta)
        print("Infilteration weigths loaded")
        #predecimos la naturaleza de los paquetes de test
        y_pred_infilt = Infilteration.predict_classes(input_N)
        print("\nInfilteration prediction: ", y_pred_infilt)
        archivoInfilt = open('InfiltOutput.txt','w')
        for element in y_pred_infilt:
            archivoInfilt.write((str)(element))
        archivoInfilt.close() #Cierras el archivo.

    elif(option==1):
        #Cargamos los resultados obtenidos en la etapa de train
        print("loading Bruteforce weigths...")
        Bruteforce.load_weights(ruta)
        print("Bruteforce weigths loaded")
        #predecimos la naturaleza de los paquetes de test
        y_pred_brute = Bruteforce.predict_classes(input_N)
        print("\nBruteforce prediction: ", y_pred_brute)
        archivoBrute = open('BruteOutput.txt','w')
        for element in y_pred_brute:
            archivoBrute.write((str)(element))
        archivoBrute.close() #Cierras el archivo.

    elif(option==2):
        #Cargamos los resultados obtenidos en la etapa de train
        print("loading DDoS weigths...")
        DDoS.load_weights(ruta)
        print("DDoS weigths loaded")
        #predecimos la naturaleza de los paquetes de test
        y_pred_dos = DDoS.predict_classes(input_N)
        print("\nDDoS prediction: ", y_pred_dos)
        archivoDoS = open('DoSOutput.txt','w')
        for element in y_pred_dos:
            archivoDoS.write((str)(element))
        archivoDoS.close() #Cierras el archivo.
```

L.3. Definición de parametros de train para medir accuracy

Código L.3: Código usado para definir los parámetros de entrenamiento de la red

```
1 # Define optimizer and objective, compile cnn
2 model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=['accuracy']) # se compila la
      red neuronal con los siguientes parametros
3 # ModelCheckpoint se usa para guardar datos o pesos en algun momento de forma que estos puedan ser
      cargados o examinados en un futuro
4 checkpointer = ModelCheckpoint(filepath="results/checkpoints/checkpoint-{epoch:02d}.hdf5",verbose=1,
      save_best_only=True,monitor='accuracy',mode='max')
5 # CSVLogger sirve para guardar los datos en un fichero CSV
6 csv_logger = CSVLogger('results/cnntrainanalysis10epochs.csv',separator=',',append=False)
7 # model.fit es la instruccion con la que se entrena la red neuronal
8 model.fit(X_train,y_train,epochs=10,callbacks=[checkpointer,csv_logger])
```

En este fragmento de código, model representa a la red. La primera instrucción (model.compile) define bajo que condiciones se quiere compilar la red. Se define el cálculo de las perdidas como categorical crossentropy A.2, el optimizador usado en el modelo como adam A.3 y las métricas a seguir como solo la exactitud del modelo. Estos parámetros son los definidos por Vinayakumar, no se ha seguido ningún desarrollo para obtener estos.

La siguiente instrucción (ModelCheckpoint) nos sirve para establecer que cada vez que se entrene el modelo(cada epoch), se evalúe si el modelo ha mejorado respecto de las anteriores iteraciones en cuanto al parámetro definido por el campo monitor. Se evalúa la mejora de la manera establecida en el campo mode. Filepath es el nombre del archivo donde se va a guardar y verbose sirve para indicar que se quiere ver una barra de progreso de cada iteración.

La tercera instrucción (CSVLogger) nos sirve para definir la ruta donde se guardan los resultados de cada iteración para una evaluación futura, en este documento csv solo se guardaran los parámetros especificados en la compilación del modelo, es decir, las perdidas y la métrica de accuracy.

Por ultimo, la ultima instrucción (model.fit) se encarga de entrenar el modelo con todos los ajustes comentados anteriormente.

L.4. Ejecución de test sobre el modelo

Model.load nos sirve para cargar los parámetros entrenados de la iteración con mejores resultados obtenidos en el paso de entrenamiento. Después, predecimos las clases de los tráficos de test. Estos datos se guardan en ficheros de texto por si se desean usar en un futuro para revisar o evaluar los resultados. Por último se evalúa el modelo comparando las salidas obtenidas de entradas de test con las salidas teóricamente correctas. Como los parámetros que se están usando para evaluar estos

Código L.4: Código usado para ejecutar un test del modelo

```
#Cargamos los resultados obtenidos en la etapa de train
model.load_weights("results/checkpoints/checkpoint-05.hdf5")

#predecimos la naturaleza de los paquetes de test
y_pred = model.predict_classes(X_test)

np.savetxt('results/expected.txt', y_test1, fmt='%01d')
np.savetxt('results/predicted.txt', y_pred, fmt='%01d')

loss, accuracy = model.evaluate(X_test, y_test)
print("\nLoss: %.2f, Accuracy: %.3f%%" %(loss, accuracy*100))
```

resultados son loss y accuracy, estos se imprime por pantalla para terminar.

M

Figuras

M.1. Muestreo sintético

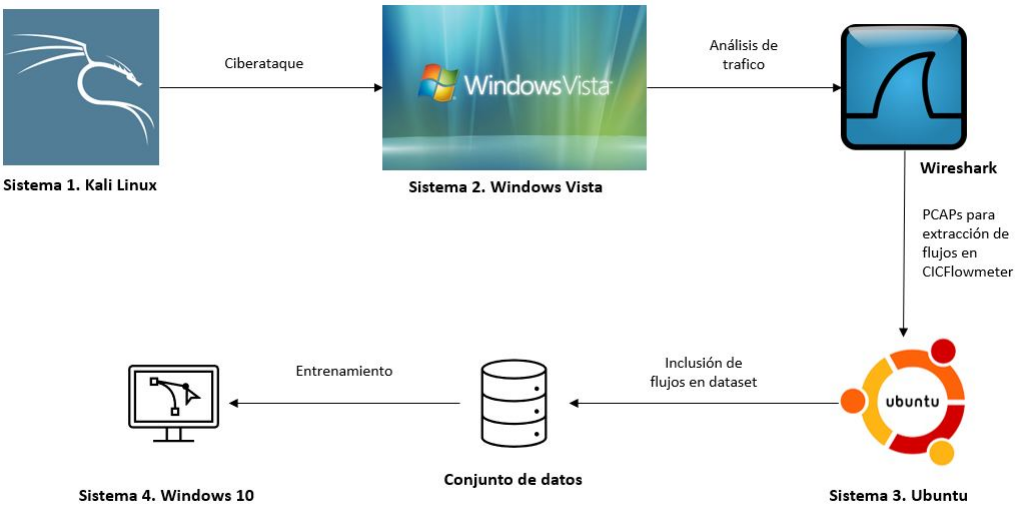

Figura M.1: Sistema para obtención de muestras sintéticas

M.2. Arquitectura paralelo CSE-CIC-IDS2018

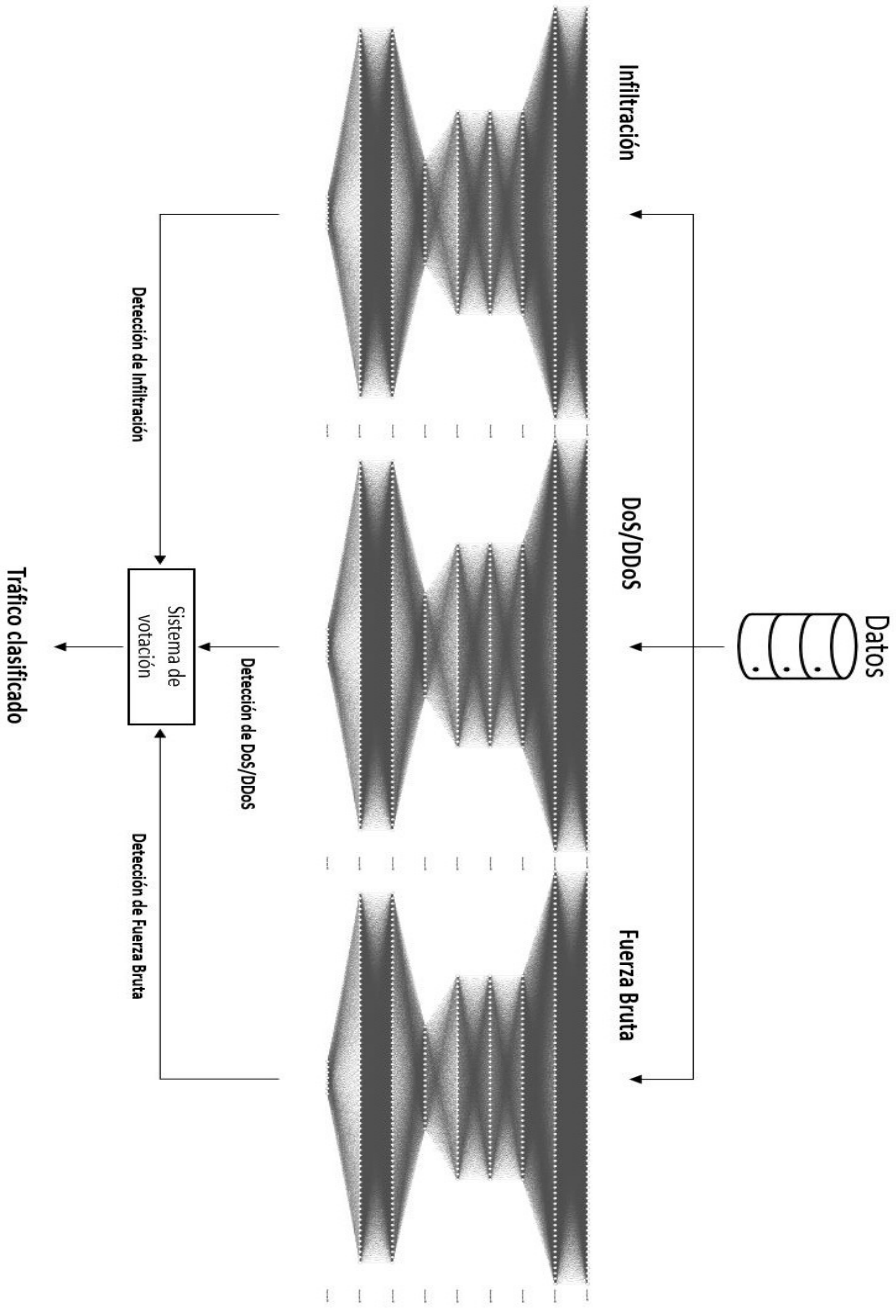

Figura M.2: Aproximación a IDS con redes en paralelo


UAM
UNIVERSIDAD AUTONOMA
DE MADRID